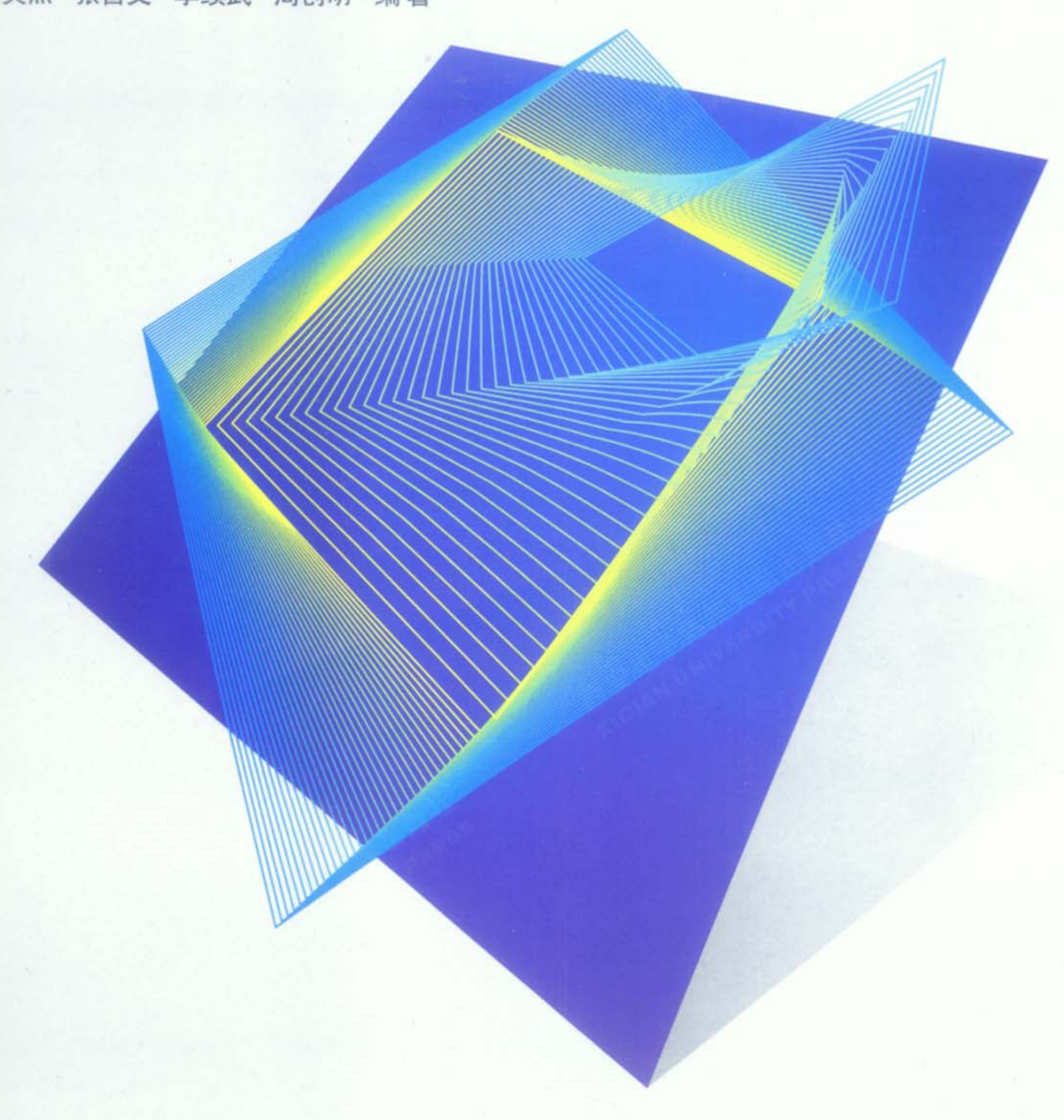
# MATLAB 遗传算法工具箱及应用

雷英杰 张善文 李续武 周创明 编著



西安电子科技大学出版社
http:// www.xduph. com

# MATLAB
# 遗传算法工具箱及应用

雷英杰　张善文
　　　　　　　　　　编著
李续武　周创明

西安电子科技大学出版社
2005

# 内 容 简 介

本书系统介绍MATLAB遗传算法和直接搜索工具箱的功能特点、编程原理及使用方法。全书共分为9章。第一章至第四章介绍遗传算法的基础知识，包括遗传算法的基本原理，编码、选择、交叉、变异，适应度函数，控制参数选择，约束条件处理，模式定理，改进的遗传算法，早熟收敛问题及其防止等。第五章至第七章介绍英国设菲尔德(Sheffield)大学的MATLAB遗传算法工具箱及其使用方法，举例说明如何利用遗传算法工具箱函数编写求解实际优化问题的MATLAB程序。第八章和第九章介绍MathWorks公司最新发布的MATLAB遗传算法与直接搜索工具箱及其使用方法。

本书取材新颖，内容丰富，逻辑严谨，语言通俗，理例结合，图文并茂，注重基础，面向应用。书中包含大量的实例，便于自学和应用。

本书可作为高等院校计算机、自动化、信息、管理、控制与系统工程等专业本科生或研究生的教材或参考书，也可供其他相关专业的师生及科研和工程技术人员自学或参考。

图书在版编目(CIP)数据
**MATLAB**遗传算法工具箱及应用/雷英杰等编著.
—西安：西安电子科技大学出版社，2005.4
ISBN 7-5606-1484-1
Ⅰ. M...　Ⅱ. 雷...　Ⅲ. 计算机辅助计算—软件包，MATLAB　Ⅳ. TP391.75
中国版本图书馆 **CIP** 数据核字(**2004**)第 **142719** 号

策　　划　戚文艳
责任编辑　龙　晖　戚文艳
出版发行　西安电子科技大学出版社(西安市太白南路2号)
电　　话　(029)88242885　88201467　　邮　编　710071
http：//www.xduph.com　　E-mail：xdupfxb@pub.xaonline.com
经　　销　新华书店
印刷单位　陕西华沐印刷科技有限责任公司
版　　次　2005年4月第1版　2005年4月第1次印刷
开　　本　787毫米×1092毫米　1/16　印张　16.875
字　　数　397千字
印　　数　1～4000册
定　　价　26.00元
ISBN 7-5606-1484-1/TP·0788(课)
**XDUP 1755001-1**

# 前　　言

MATLAB是MathWorks公司推出的一套高性能的数值计算和可视化软件。它集数值分析、矩阵运算、信号处理和图形显示于一体，构成一个方便的、界面友好的用户环境。MATLAB强大的扩展功能和影响力吸引各个领域的专家相继推出了许多基于MATLAB的专用工具箱。MATLAB强大的科学运算、灵活的程序设计流程、高质量的图形可视化与界面设计、便捷的与其他程序和语言的接口等功能，使之成为当今世界最有活力和最具影响力的可视化软件。

遗传算法(Genetic Algorithm，简称GA)是以自然选择和遗传理论为基础，将生物进化过程中适者生存规则与群体内部染色体的随机信息交换机制相结合的高效全局寻优搜索算法。GA摒弃了传统的搜索方式，模拟自然界生物进化过程，采用人工进化的方式对目标空间进行随机优化搜索。它将问题域中的可能解看做是群体的一个个体或染色体，并将每一个个体编码成符号串形式，模拟达尔文的遗传选择和自然淘汰的生物进化过程，对群体反复进行基于遗传学的操作(遗传、交叉和变异)。根据预定的目标适应度函数对每个个体进行评价，依据适者生存、优胜劣汰的进化规则，不断得到更优的群体，同时以全局并行搜索方式来搜索优化群体中的最优个体，以求得满足要求的最优解。

自从1975年John H. Holland教授出版关于GA的经典之作《Adaptation in Natural and Artificial Systems》以来，GA已获得广泛应用。从遗传算法的整个发展来看，20世纪70年代是兴起阶段，20世纪80年代是发展阶段，20世纪90年代是高潮阶段。遗传算法作为一种实用、高效、鲁棒性强的优化技术，发展极为迅速，已引起国内外学者的高度重视。

遗传算法提供了一种求解非线性、多模型、多目标等复杂系统优化问题的通用框架，它不依赖于问题具体的领域，已经广泛应用于函数优化、组合优化、自动控制、机器人学、图像处理、人工生命、遗传编码、机器学习等科技领域，并且已经在求解旅行商问题、背包问题、装箱问题、图形划分问题等方面的应用取得了成功。

由于GA在大量问题求解过程中独特的优点和广泛的应用，许多基于MATLAB的遗传算法工具箱相继出现，其中出现较早、影响较大、较为完备者当属英国设菲尔德(Sheffield)大学推出的基于MATLAB的遗传算法工具箱。另外，还有美国北卡罗莱纳那州立大学推出的可与MATLAB一起使用的遗传算法优化工具箱GAOT(Genetic Algorithm Optimization Toolbox)。考虑到前者在内容上已经覆盖到后者，因此本书将着重介绍英国设菲尔德大学的基于MATLAB的遗传算法工具箱。值得注意的是，MathWorks公司最新发布了一个专门设计的MATLAB遗传算法与直接搜索工具箱(Genetic Algorithm and Direct Search Toolbox)，本书同时也详细介绍了这个遗传算法与直接搜索工具箱及其使用方法。

书中通过大量实例，介绍了如何利用提供的遗传算法工具箱函数来编写MATLAB程序，解决实际问题。

在此，作者非常感谢西安电子科技大学王宝树教授、周利华教授、李荣才教授等的指导和鼓励，以及空军工程大学计算机系吕辉教授等的支持和帮助，真诚感谢西安电子科技大学出版社的大力支持。

需要特别指出，虽然作者竭尽所能，精心策划章节结构和内容编排，详细测试书中的每一个实例，尽可能简明而准确地表述其意，但限于水平和资料，书中的错误和不足之处在所难免，恳请读者不吝指正。

作　者

2004年10月

# 目　录

# 第一章　遗传算法概述

遗传算法(Genetic Algorithm，简称 GA)起源于对生物系统所进行的计算机模拟研究。美国 Michigan 大学的 Holland 教授及其学生受到生物模拟技术的启发，创造出了一种基于生物遗传和进化机制的适合于复杂系统优化的自适应概率优化技术——遗传算法。1967 年，Holland 的学生 Bagley 在其博士论文中首次提出了“遗传算法”一词，他发展了复制、交叉、变异、显性、倒位等遗传算子，在个体编码上使用双倍体的编码方法。Holland 教授用遗传算法的思想对自然和人工自适应系统进行了研究，提出了遗传算法的基本定理——模式定理(Schema Theorem)，并于 1975 年出版了第一本系统论述遗传算法和人工自适应系统的专著《Adaptation in Natural and Artificial Systems》。20 世纪 80 年代，Holland 教授实现了第一个基于遗传算法的机器学习系统，开创了遗传算法的机器学习的新概念。1975 年，De Jong 基于遗传算法的思想在计算机上进行了大量的纯数值函数优化计算实验，建立了遗传算法的工作框架，得到了一些重要且具有指导意义的结论。1989 年，Goldberg 出版了《Genetic Algorithm in Search，Optimization and Machine Learning》一书，系统地总结了遗传算法的主要研究成果，全面完整地论述了遗传算法的基本原理及其应用。1991 年，Davis 出版了《Handbook of Genetic Algorithms》一书，介绍了遗传算法在科学计算、工程技术和社会经济中的大量实例。1992 年，Koza 将遗传算法应用于计算机程序的优化设计及自动生成，提出了遗传编程(Genetic Programming，简称 GP)的概念。在控制系统的离线设计方面遗传算法被众多的使用者证明是有效的策略。例如，Krishnakumar 和 Goldberg 以及 Bramlette 和 Cusin 已证明使用遗传优化方法在太空应用中导出优异的控制器结构比使用传统方法如 LQR 和 Powell(鲍威尔)的增音机设计所用的时间要少(功能评估)。Porter 和 Mohamed 展示了使用本质结构分派任务的多变量飞行控制系统的遗传设计方案。与此同时，另一些人证明了遗传算法如何在控制器结构的选择中使用。

从遗传算法的整个发展过程来看，20 世纪 70 年代是兴起阶段，20 世纪 80 年代是发展阶段，20 世纪 90 年代是高潮阶段。遗传算法作为一种实用、高效、鲁棒性强的优化技术，发展极为迅速，已引起国内外学者的高度重视。

## 1.1　遗传算法的概念

生物的进化(Evolution)过程主要是通过染色体之间的交叉和变异来完成的。基于对自然界中生物遗传与进化机理的模仿，针对不同的问题，很多学者设计了许多不同的编码方

法来表示问题的可行解，开发出了许多种不同的编码方式来模仿不同环境下的生物遗传特性。这样，由不同的编码(Coding)方法和不同的遗传算子就构成了各种不同的遗传算法。

遗传算法是模仿自然界生物进化机制发展起来的随机全局搜索和优化方法，它借鉴了达尔文的进化论和孟德尔的遗传学说。其本质是一种高效、并行、全局搜索的方法，它能在搜索过程中自动获取和积累有关搜索空间的知识，并自适应地控制搜索过程以求得最优解。遗传算法操作使用适者生存的原则，在潜在的解决方案种群中逐次产生一个近似最优的方案。在遗传算法的每一代中，根据个体在问题域中的适应度值和从自然遗传学中借鉴来的再造方法进行个体选择，产生一个新的近似解。这个过程导致种群中个体的进化，得到的新个体比原个体更能适应环境，就像自然界中的改造一样。

个体或当前近似解被编码为由字母组成的串，即染色体(Chromosome)，使基因(Gene，染色体值）能在(表现）域决策变量上被惟一地描述。尽管可以使用二进制、整数、实值等，但是在遗传算法表现型上最常用的仍是二进制字符串。例如，一个问题具有两个变量 X1 和 X2，它们的染色体结构能用图 1.1 所示的方法描述。




图 1.1　个体的染色体结构

X1 被编码为 10 位，X2 被编码为 15 位(位数的多少能够反映精确度水平或个体决策变量的范围)，是一个不含人们试图解决问题的信息的染色体串。这只是表现值的染色体编码，任何意义均可应用于表现型。无论如何，就像下面的描述，搜索过程将在决策变量的编码中而不是它们自身中操作，当然除了在实值基因被使用的情况。

在决策变量域中的染色体表现型已被编码，可以估计种群的个体成员的特性或适应度。通过特征目标函数来估计个体在问题域中的特性。在自然世界中，这就是个体在现行环境中的生存能力。因此，目标函数建立的基础是在整个繁殖过程中选择成对的个体进行交配。

在再生(复制）期间，每个个体均被计算适应度值，它来自没有加工的原始特性度量，由目标函数给出。这个值用来在选择中偏向更加适合的个体。相对整个种群，适应度高的个体具有高的选中参加交配的概率，而适应度低的个体具有相对低的选中概率。

一旦个体计算了适应度值，个体能根据它们的相对适应度从种群中被选中并重组，产生下一代。遗传算子直接操作染色体的特征(基因)，使用一般情况下个体的基因代码，产生更适合的个体。重组算子用在一对个体或一大组个体中交换基因信息。最简单的重组算子是单点交叉。

考虑两个二进制父代串：

$$A=10010110 \quad 和 \quad B=10111000$$

如果一个整 $I$ 是随机地在 1 到串长 $L$ 减 1 之间(即$[1, L-1]$)选择的，在这点后，两个个体间的基因进行交换，随后两个子代串产生。例如，当交叉点 $I=5$ 时，两个子代串产生如下：

$$A'=10010000 \quad 和 \quad B'=10111110$$

交叉算子并不是必须在种群的所有串中执行的。当一对个体被选中培育下一代时，代

替的是应用一个概率 $P_x$。进一步的遗传算法称为变异，再次使用一个概率 $P_m$ 应用到新染色体上。变异能根据一些概率准则引起个体基因表现型发生变化，在二进制表现型中，变异引起单个位的状态变化，即 0 变 1，或者 1 变 0。

例如，在 A′中变异第四位，1 变 0，产生新串为 10000000。

变异通常被认为是一后台算子，以确保研究问题空间的特殊子空间的概率永不为 0。变异具有阻止局部最优收敛的作用。

在重组和变异后，如果需要，这些个体串随后被解码，进行目标函数评估，计算每个个体的适应度值，个体根据适应度被选择参加交配，并且这个过程继续直到产生子代。在这种方法中，种群中个体的平均性能希望得到提高，好的个体被保存并且相互产生下一代，而低适应度的个体则消失。当一些判定条件满足后，遗传算法则终止，例如，一定的遗传代数、种群的均差或遇到搜索空间的特殊点。

## 1.2　遗传算法的特点

遗传算法是一种借鉴生物界自然选择(Natural Selection)和自然遗传机制的随机搜索算法(Random Searching Algorithms)。它与传统的算法不同，大多数古典的优化算法是基于一个单一的度量函数(评估函数)的梯度或较高次统计，以产生一个确定性的试验解序列；遗传算法不依赖于梯度信息，而是通过模拟自然进化过程来搜索最优解(Optimal Solution)，它利用某种编码技术，作用于称为染色体的数字串，模拟由这些串组成的群体的进化过程。遗传算法通过有组织的、随机的信息交换来重新组合那些适应性好的串，生成新的串的群体。

### 1.2.1　遗传算法的优点

遗传算法具有如下优点：

(1) 对可行解表示的广泛性。遗传算法的处理对象不是参数本身，而是针对那些通过参数集进行编码得到的基因个体。此编码操作使得遗传算法可以直接对结构对象进行操作。所谓结构对象，泛指集合、序列、矩阵、树、图、链和表等各种一维或二维甚至多维结构形式的对象。这一特点使得遗传算法具有广泛的应用领域。比如：

① 通过对连接矩阵的操作，遗传算法可用来对神经网络或自动机的结构或参数加以优化。

② 通过对集合的操作，遗传算法可实现对规则集合和知识库的精炼而达到高质量的机器学习目的。

③ 通过对树结构的操作，用遗传算法可得到用于分类的最佳决策树。

④ 通过对任务序列的操作，遗传算法可用于任务规划，而通过对操作序列的处理，可自动构造顺序控制系统。

(2) 群体搜索特性。许多传统的搜索方法都是单点搜索，这种点对点的搜索方法，对于多峰分布的搜索空间常常会陷于局部的某个单峰的极值点。相反，遗传算法采用的是同时处理群体中多个个体的方法，即同时对搜索空间中的多个解进行评估。这一特点使遗传算法具有较好的全局搜索性能，也使得遗传算法本身易于并行化。

(3) 不需要辅助信息。遗传算法仅用适应度函数的数值来评估基因个体，并在此基础上进行遗传操作。更重要的是，遗传算法的适应度函数不仅不受连续可微的约束，而且其定义域可以任意设定。对适应度函数的惟一要求是，编码必须与可行解空间对应，不能有死码。由于限制条件的缩小，使得遗传算法的应用范围大大扩展。

(4) 内在启发式随机搜索特性。遗传算法不是采用确定性规则，而是采用概率的变迁规则来指导它的搜索方向。概率仅仅是作为一种工具来引导其搜索过程朝着搜索空间的更优化的解区域移动的。虽然看起来它是一种盲目搜索方法，实际上它有明确的搜索方向，具有内在的并行搜索机制。

(5) 遗传算法在搜索过程中不容易陷入局部最优，即使在所定义的适应度函数是不连续的、非规则的或有噪声的情况下，也能以很大的概率找到全局最优解。

(6) 遗传算法采用自然进化机制来表现复杂的现象，能够快速可靠地解决求解非常困难的问题。

(7) 遗传算法具有固有的并行性和并行计算的能力。

(8) 遗传算法具有可扩展性，易于同别的技术混合。

应重点注意的是，遗传算法对给定问题给出了大量可能的解答，并挑选最终的解答给用户。要是一个特定问题并没有单个的解，例如 Pareto 最优解系列中，就像多目标优化和日程安排的案例中，遗传算法将尽可能地用于识别可同时替换的解。

### 1.2.2 遗传算法的不足之处

遗传算法作为一种优化方法，它存在自身的局限性：

(1) 编码不规范及编码存在表示的不准确性。

(2) 单一的遗传算法编码不能全面地将优化问题的约束表示出来。考虑约束的一个方法就是对不可行解采用阈值，这样，计算的时间必然增加。

(3) 遗传算法通常的效率比其他传统的优化方法低。

(4) 遗传算法容易出现过早收敛。

(5) 遗传算法对算法的精度、可信度、计算复杂性等方面，还没有有效的定量分析方法。

## 1.3 遗传算法与传统方法的比较

对于一个求函数最大值的优化问题(求函数最小值也类似)，一般可描述为带约束条件的数学规划模型：

$$\begin{cases} \max & f(X) \\ \text{s.t.} & X \in R \\ & R \subseteq U \end{cases} \tag{1.1}$$

式中，$X=[x_1, x_2, \cdots, x_n]^{\mathrm{T}}$ 为决策变量，$f(X)$ 为目标函数，$U$ 为基本空间，$R$ 是 $U$ 的一个子集。满足约束条件的解称为可行解(Feasible Solution)，集合 $R$ 表示由所有满足约束条件的解所组成的一个集合，称为可行解集合。它们之间的关系如图 1.2 所示。

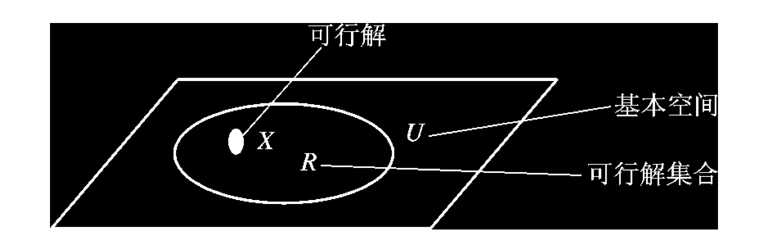



图 1.2　最优化问题的可行解及可行解集合

对于上述最优化问题，目标函数和约束条件种类繁多，有的是线性的，有的是非线性的；有的是连续的，有的是离散的；有的是单峰值的，有的是多峰值的。随着研究的深入，人们逐渐认识到在很多复杂情况下要想完全精确地求出其最优解是不可能的，也是不现实的，因而求出其近似最优解或满意解是人们主要研究的问题之一。

对于类似上述最优化问题，求最优解或近似最优解的传统方法主要有解析法、随机法和穷举法。解析法主要包括爬山法和间接法。随机法主要包括导向随机方法和盲目随机方法。而穷举法主要包括完全穷举法、回溯法、动态规划法和限界剪枝法。

此类问题可以利用遗传算法求解。而对于求解此类问题，遗传算法与一般传统方法有着本质的区别。图 1.3 所示为传统算法和遗传算法对比示意图。

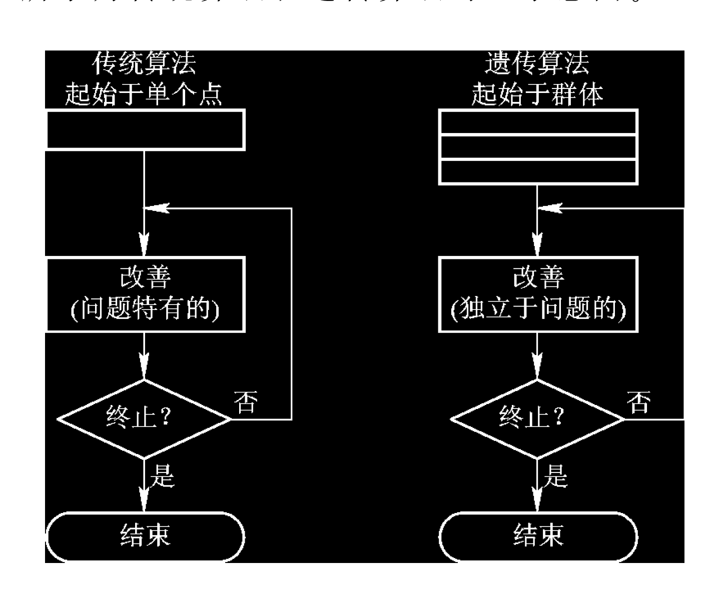



图 1.3　传统算法和遗传算法对比

**1．**遗传算法与启发式算法的比较

启发式算法是指通过寻求一种能产生可行解的启发式规则，找到问题的一个最优解或近似最优解。该方法求解问题的效率较高，但是它对每一个所求的问题必须找出其特有的启发式规则，这个启发式规则一般无通用性，不适用于其他问题。但遗传算法采用的不是确定性规则，而是强调利用概率转换规则来引导搜索过程。

**2．**遗传算法与爬山法的比较

爬山法是直接法、梯度法和 Hessian 法的通称。爬山法首先在最优解可能存在的地方选择一个初始点，然后通过分析目标函数的特性，由初始点移到一个新的点，然后再继续这个过程。爬山法的搜索过程是确定的，它通过产生一系列的点收敛到最优解(有时是局部最优解)，而遗传算法的搜索过程是随机的，它产生一系列随机种群序列。二者的主要差异可以归纳为如下两点：

(1) 爬山法的初始点仅有一个，由决策者给出；遗传算法的初始点有多个，是随机产

生的。

（2）通过分析目标函数的特性可知，爬山法由上一点产生一个新的点；遗传算法通过遗传操作，在当前的种群中经过交叉、变异和选择产生下一代种群。对同一优化问题，遗传算法所使用的机时比爬山法所花费的机时要多。但遗传算法可以处理一些爬山法所不能解决的复杂的优化问题。

**3．** 遗传算法与穷举法的比较

穷举法就是对解空间内的所有可行解进行搜索，但是通常的穷举法并不是完全穷举法，即不是对所有解进行尝试，而是有选择地尝试，如动态规划法、限界剪枝法。对于特定的问题，穷举法有时也表现出很好的性能。但一般情况下，对于完全穷举法，方法简单易行，但求解效率太低；对于动态规划法、限界剪枝法，则鲁棒性不强。相比较而言，遗传算法具有较高的搜索能力和极强的鲁棒性。

**4．** 遗传算法与盲目随机法的比较

与上述的搜索方法相比，盲目随机搜索法有所改进，但它的搜索效率仍然不高。一般而言，只有解在搜索空间中形成紧致分布时，它的搜索才有效。而遗传算法作为导向随机搜索方法，是对一个被编码的参数空间进行高效搜索。

经过上面的探讨，能看到遗传算法与更多的传统优化方法在本质上有着不同之处，主要有四点不同：

（1）遗传算法搜索种群中的点是并行的，而不是单点。

（2）遗传算法并不需要辅助信息或辅助知识，只需要影响搜索方向的目标函数和相对应的适应度。

（3）遗传算法使用概率变换规则，而不是确定的变换规则。

（4）遗传算法工作使用编码参数集，而不是自身的参数集(除了在实值个体中使用)。

## 1.4　遗传算法的基本用语

由于遗传算法是自然遗传学和计算机科学相互结合渗透而成的新的计算方法，因此遗传算法中经常使用自然进化中有关的一些基本用语。了解这些用语对于讨论和应用遗传算法是十分必要的。

生物的遗传物质的主要载体是染色体，DNA 是其中最主要的遗传物质，而基因(Gene)又是控制生物性状的遗传物质的功能单位和结合单位。复数个基因组成染色体，染色体中基因的位置称为基因座(Locus)，而基因所取的值叫做等位基因(Allele)。基因和基因座决定了染色体的特征，也就决定了生物个体的性质状态。染色体有两种相应的表示模式，即基因型和表现型。所谓表现型，是指生物个体所表现出来的性质状态，而基因型是指与表现型密切相关的基因组成。同一种基因型的生物个体在不同的环境条件下可以有不同的表现型，因此表现型是基因型与环境条件相互作用的结果。在遗传算法中，染色体对应的是数据或数组，在标准的遗传算法中，通常是由一维的串结构数据表现的。串上各个位置对应上述的基因座，而各位置上所取的值对应上述的等位基因。遗传算法处理的是染色体，或者叫基因型个体。一定数量的个体组成了群体，也叫集团。群体中个体的数目称为群体的大小，也叫群体规模。而各个体对环境的适应程度叫适应度。执行遗传算法时包

含两个必要的数据转换操作，一个是表现型到基因型的转换，它把搜索空间中的参数或解转换成遗传空间中的染色体或个体，此过程称为编码操作；另一个是基因型到表现型的转换，它是前者的一个相反操作，称为译码操作。表 1.1 为自然遗传学和人工遗传算法中所使用的基本用语的对应关系。

表 **1.1** 遗传学和遗传算法中基本用语对照表

| 自然遗传算法 | 人工遗传算法 |
| --- | --- |
| 染色体(Chromosome) | 解的编码(数据、数组、位串) |
| 基因(Gene) | 解中每一分量的特征(特性、个性、探测器、位) |
| 等位基因(Allele) | 特性值 |
| 基因座(Locus) | 串中位置 |
| 基因型(Genptype) | 结构 |
| 表现型(Phenotype) | 参数集、解码结构、候选解 |
| 遗传隐匿 | 非线性 |
| 个体(Individual) | 解 |
| 适者生存 | 在算法停止时，最优目标值的解有最大的可能被留住 |
| 适应性(Fitness) | 适应度函数值 |
| 群体(Population) | 选定的一组解(其中解的个数为群体的规模) |
| 复制(Reproduction) | 根据适应函数值选取的一组解 |
| 交配(Crossover) | 通过交配原则产生一组新解的过程 |
| 变异(Mutation) | 编码的某一个分量发生变化的过程 |

## 1.5 遗传算法的研究方向

遗传算法是多学科结合与渗透的产物，它已经发展成一种自组织、自适应的综合技术，其研究方向主要有下述几个方面。

**1.** 基础理论

遗传算法的数学理论并不完善，张铃等对遗传算法的“模式定理”和“隐性并行性”进行了分析研究，指出其不足并指出遗传算法本质上是一个具有定向制导的随机搜索技术。在遗传算法中，群体规模和遗传算子的控制参数的选取非常困难，但它们又是必不可少的实验参数，在这方面，已有一些具有指导性的实验结果。遗传算法还有一个过早收敛的问题，如何阻止过早收敛也是人们正在研究的问题之一。

**2. 分布并行遗传算法**

遗传算法在操作上具有高度的并行性，许多研究人员都在探索在并行机和分布式系统上高效执行遗传算法的策略。对分布遗传算法的研究表明，只要通过保持多个群体和恰当控制群体间的相互作用来模拟并发执行过程，即使不使用并行计算机，也能提高算法的执行效率。遗传算法的并行性主要从三个方面考虑，即个体适应度评价的并行性、整个群体各个个体适应度评价的并行性及子代群体产生过程的并行性。

**3. 分类系统**

分类系统属于基于遗传算法的机器学习中的一类，包括一个简单的基于串规则的并行生成子系统、规则评价子系统和遗传算法子系统。分类系统被人们越来越多地应用在科学、工程和经济领域中，是目前遗传算法研究中一个十分活跃的方向。

**4. 遗传神经网络**

遗传神经网络包括连接级、网络结构和学习规则的进化。遗传算法与神经网络相结合，成功地用于从分析时间序列来进行财政预算。在这些系统中，训练信号是模糊的，数据是有噪声的，一般很难正确给出每个执行的定量评价，如果采用遗传算法学习，就能克服这些困难，显著提高系统性能。Muhlenbein 分析了多层感知网络的局限性，并猜想下一代神经网络将是遗传神经网络。

**5. 进化算法**

模拟自然进化过程可以产生鲁棒的计算机算法——进化算法。遗传算法是其三种典型的算法之一，其余两种算法是进化规划和进化策略，这三种算法是独立发展起来的。

**6. 人工生命与遗传算法**

近几年来，通过计算机模拟再现种种生命现象，以达到对生命更深刻理解的人工生命的研究正在兴起。已有不少学者对生态系统的演变、食物链的维持以及免疫系统的进化等用遗传算法做了生动的模拟。但是实现人工生命的手段很多，遗传算法在实现人工生命中的基本地位和能力究竟如何，这是值得研究的课题。

## 1.6 基于遗传算法的应用

遗传算法提供了一种求解复杂系统优化问题的通用框架，它不依赖于问题具体的领域，对问题的种类有很强的鲁棒性，所以广泛应用于许多学科。近十年来，遗传算法得到了迅速发展。目前，遗传算法在生物技术和生物学、化学和化学工程、计算机辅助设计、物理学和数据分析、动态处理、建模与模拟、医学与医学工程、微电子学、模式识别、人工智能、生产调度、机器人学、开矿工程、电信学、售货服务系统等领域都得到应用，成为求解全局优化问题的有力工具之一。下面列出遗传算法一些主要的应用领域。

**1. 函数优化**

函数优化(Function Optimization)是遗传算法的经典应用领域，也是对遗传算法进行性能评价的常用算例。可以用各种各样的函数来验证遗传算法的性能。对一些非线性、多模型、多目标的函数优化问题，使用遗传算法可得到较好的结果。

**2. 组合优化**

随着问题规模的增大，组合优化问题的搜索空间也急剧扩大，有时在目前的计算机上

用枚举法很难甚至不能求出问题的最优解，对这类问题，人们已意识到应把主要精力放在寻求其满意解上，而遗传算法就是寻求这种满意解的最佳工具之一。实践证明，遗传算法对于组合优化中的NP完全问题非常有效。

**3．** 生产调度问题

采用遗传算法能够解决复杂的生产调度问题。在单件生产车间调度、流水线生产车间调度、生产规划、任务分配等方面，遗传算法都得到了有效的应用。

**4．** 自动控制

在自动控制领域中有很多与优化相关的问题需要求解，遗传算法已在其中得到了初步应用，并显示出了良好的效果。例如，基于遗传算法的模糊控制器优化设计，用遗传算法进行航空控制系统的优化，使用遗传算法设计空间交会控制器等。

**5．** 机器人学

机器人是一类复杂的难以精确建模的人工系统，而遗传算法的起源来自于对人工自适应系统的研究，所以机器人学理所当然地成为遗传算法的一个重要领域。例如，遗传算法已经在移动机器人路径规划、机器人逆运动学求解等方面得到很好的应用。

**6．** 图像处理

图像处理是计算机视觉中的一个重要领域，在图像处理过程中，如扫描、特征提取、图像分割等不可避免地会存在一些误差，这些误差会影响图像处理的效果。如何使这些误差最小是使计算机视觉达到实用化的重要要求，遗传算法在这些图像处理的优化计算方面找到了用武之地。

**7．** 遗传编程

Koza发展了遗传程序设计的概念，他使用了以LISP语言所表示的编码方法，算法基于对一种树型结构所进行的遗传操作来自动生成计算机程序。

**8．** 机器学习

基于遗传算法的机器学习，特别是分类器系统，在很多领域中都得到了应用。例如，遗传算法被用于学习模糊控制规则，利用遗传算法来学习隶属函数等。基于遗传算法的机器学习可用于调整人工神经网络的连接权，也可用于神经网络结构的优化设计。分类器系统在多机器人路径规划系统中得到了成功的应用。

**9．** 数据挖掘

数据挖掘(Data Mining)是指从大型数据库或数据仓库中提取隐含的、未知的、非平凡的及有潜在应用价值的信息或模式，它是数据库研究中的一个很有应用价值的新领域，由于遗传算法的特点，遗传算法可用于数据挖掘中的规则开采。

**10．** 信息战

遗传算法在信息战领域得到了初步应用。使用遗传算法能够进行雷达目标识别、数据挖掘、作战仿真、雷达辐射源识别、雷达天线优化设计、雷达目标跟踪、盲信号处理、空间谱估计、天线设计、网络入侵检测、情报分析中的数据挖掘和数据融合、信息战系统仿真、作战效能评估、作战辅助决策等。

表1.2列出了遗传算法的主要应用领域及问题举例。

表 **1.2** 遗传算法的主要应用领域

| 应用领域 | 例　　子 |
| --- | --- |
| 控制 | 瓦斯管道控制，防避导弹控制，机器人控制 |
| 规划 | 生产规划，并行机任务分配 |
| 设计 | VLSI 布局，通信网络设计，喷气发动机设计 |
| 组合优化 | TSP 问题，背包问题，图划分问题 |
| 图像处理 | 模式识别，特征提取，图像恢复 |
| 信号处理 | 滤波器设计，目标识别，运动目标分割 |
| 机器人 | 路径规划 |
| 人工生命 | 生命的遗传进化 |

# 第二章　基本遗传算法及改进

Holland创建的遗传算法是一种概率搜索算法，它利用某种编码技术作用于称为染色体的数串，其基本思想是模拟由这些串组成的个体进化过程。该算法通过有组织的、然而是随机的信息交换，重新组合那些适应性好的串。在每一代中，利用上一代串结构中适应好的位和段来生成一个新的串的群体；作为额外增添，偶尔也要在串结构中尝试用新的位和段来替代原来的部分。

遗传算法是一类随机优化算法，它可以有效地利用已有的信息处理来搜索那些有希望改善解质量的串。类似于自然进化，遗传算法通过作用于染色体上的基因，寻找好的染色体来求解问题。与自然界相似，遗传算法对待求解问题本身一无所知，它所需要的仅是对算法所产生的每个染色体进行评价，并基于适应度值来改变染色体，使适用性好的染色体比适应性差的染色体有更多的繁殖机会。

## 2.1　遗传算法的运行过程

遗传算法模拟了自然选择和遗传中发生的复制、交叉和变异等现象，从任一初始种群(Population)出发，通过随机选择、交叉和变异操作，产生一群更适应环境的个体，使群体进化到搜索空间中越来越好的区域，这样一代一代地不断繁衍进化，最后收敛到一群最适应环境的个体(Individual)，求得问题的最优解。

### 2.1.1　完整的遗传算法运算流程

遗传算法的一般步骤如图2.1所示。

完整的遗传算法运算流程可以用图2.2来描述。

由图2.2可以看出，使用上述三种遗传算子(选择算子、交叉算子和变异算子)的遗传算法的主要运算过程如下：

(1) 编码：解空间中的解数据 $x$，作为遗传算法的表现型形式。从表现型到基因型的映射称为编码。遗传算法在进行搜索之前先将解空间的解数据表示成遗传空间的基因型串结构数据，这些串结构数据的不同组合就构成了不同的点。

(2) 初始群体的生成：随机产生 $N$ 个初始串结构数据，每个串结构数据称为一个个体，$N$ 个个体构成了一个群体。遗传算法以这 $N$ 个串结构作为初始点开始迭代。设置进化代数计数器 $t \leftarrow 0$；设置最大进化代数 $T$；随机生成 $M$ 个个体作为初始群体 $P(0)$。

(3) 适应度值评价检测：适应度函数表明个体或解的优劣性。对于不同的问题，适应度函数的定义方式不同。根据具体问题，计算群体 $P(t)$ 中各个个体的适应度。

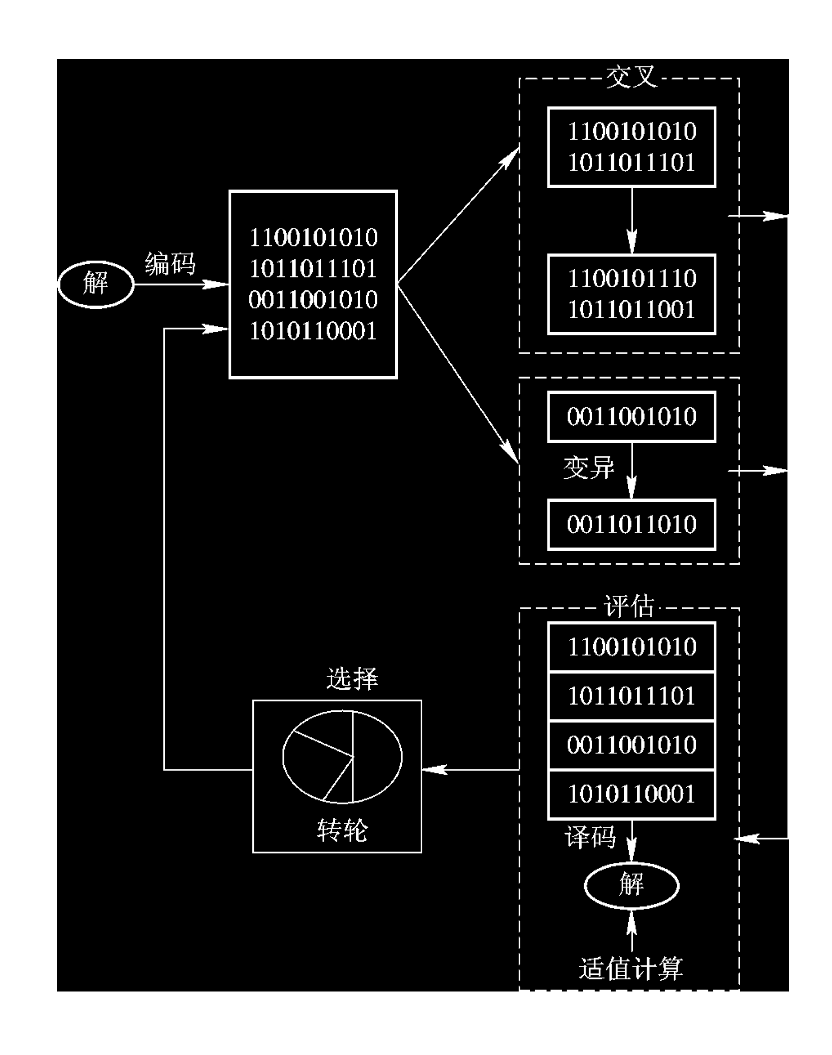



图 2.1　遗传算法的一般步骤

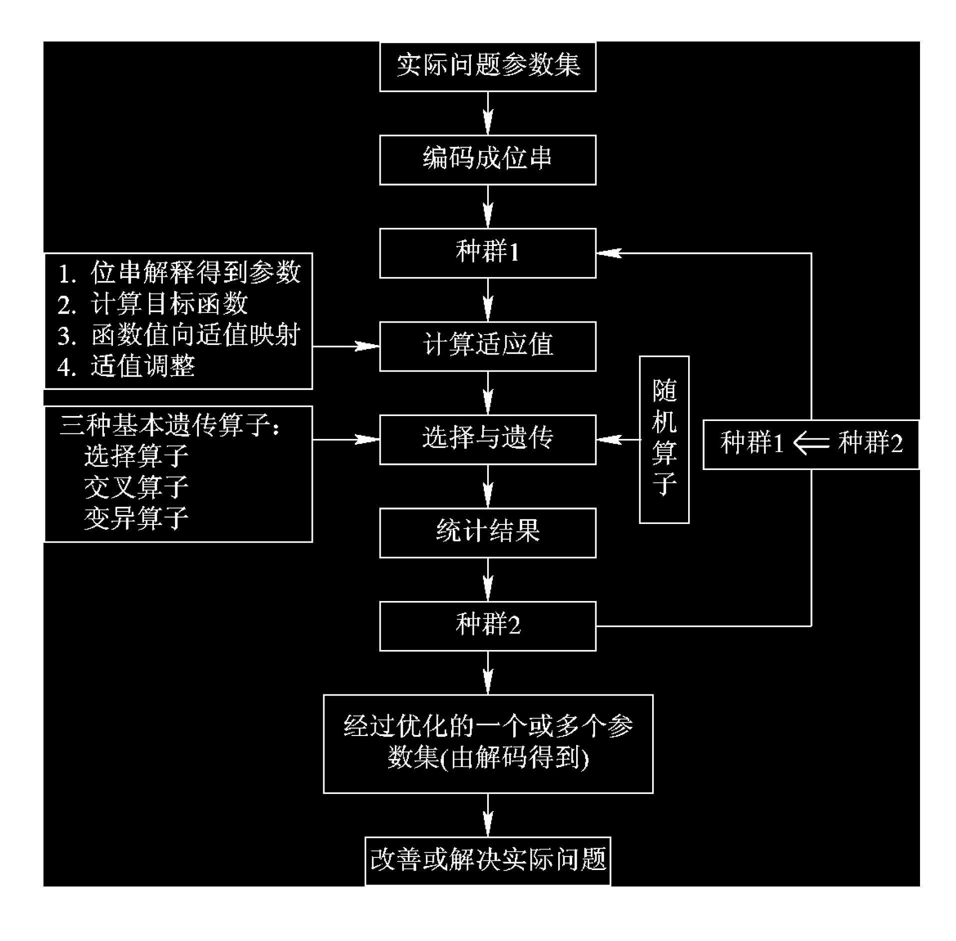



图 2.2　遗传算法运算流程

（4）选择：将选择算子作用于群体。

（5）交叉：将交叉算子作用于群体。

（6）变异：将变异算子作用于群体。

群体 $P(t)$ 经过选择、交叉、变异运算后得到下一代群体 $P(t+1)$。

（7）终止条件判断：若 $t \leqslant T$，则 $t \leftarrow t+1$，转到步骤（2）；若 $t > T$，则以进化过程中所得到的具有最大适应度的个体作为最优解输出，终止运算。

从遗传算法运算流程可以看出，进化操作过程简单，容易理解，它给其他各种遗传算法提供了一个基本框架。

一个简单的遗传算法被 Goldberg 用来进行轮廓描述并用来举例说明遗传算法的基本组成。$t$ 代种群用变量 $P(t)$ 表示，初始种群 $P(0)$ 是随机设计的，简单遗传算法的伪代码描述如下：

```
procedure GA
begin
    t=0;
    initialize P(t);
    evaluate P(t);
    while not finished do
    begin
        t=t+1;
        select P(t) from P(t-1);
        reproduce pairs in P(t);
        evaluate P(t);
    end
end
```

### 2.1.2 遗传算法的基本操作

遗传算法有三个基本操作：选择（Selection）、交叉（Crossover）和变异（Mutation）。

（1）选择。选择的目的是为了从当前群体中选出优良的个体，使它们有机会作为父代为下一代繁殖子孙。根据各个个体的适应度值，按照一定的规则或方法从上一代群体中选择出一些优良的个体遗传到下一代群体中。遗传算法通过选择运算体现这一思想，进行选择的原则是适应性强的个体为下一代贡献一个或多个后代的概率大。这样就体现了达尔文的适者生存原则。

（2）交叉。交叉操作是遗传算法中最主要的遗传操作。通过交叉操作可以得到新一代个体，新个体组合了父辈个体的特性。将群体内的各个个体随机搭配成对，对每一个个体，以某个概率（称为交叉概率，Crossover Rate）交换它们之间的部分染色体。交叉体现了信息交换的思想。

（3）变异。变异操作首先在群体中随机选择一个个体，对于选中的个体以一定的概率随机改变串结构数据中某个串的值，即对群体中的每一个个体，以某一概率（称为变异概率，Mutation Rate）改变某一个或某一些基因座上的基因值为其他的等位基因。同生物界一样，遗传算法中变异发生的概率很低。变异为新个体的产生提供了机会。

## 2.2 基本遗传算法

基本遗传算法(也称标准遗传算法或简单遗传算法，Simple Genetic Algorithm，简称SGA)是一种群体型操作，该操作以群体中的所有个体为对象，只使用基本遗传算子(Genetic Operator)：选择算子(Selection Operator)、交叉算子(Crossover Operator)和变异算子(Mutation Operator)，其遗传进化操作过程简单，容易理解，是其他一些遗传算法的基础，它不仅给各种遗传算法提供了一个基本框架，同时也具有一定的应用价值。选择、交叉和变异是遗传算法的3个主要操作算子，它们构成了所谓的遗传操作，使遗传算法具有了其他传统方法没有的特点。

### 2.2.1 基本遗传算法的数学模型

基本遗传算法可表示为

$$SGA = (C, E, P_0, M, \Phi, \Gamma, \Psi, T) \tag{2.1}$$

式中：$C$——个体的编码方法；

$E$——个体适应度评价函数；

$P_0$——初始种群；

$M$——种群大小；

$\Phi$——选择算子；

$\Gamma$——交叉算子；

$\Psi$——变异算子；

$T$——遗传运算终止条件。

图2.3所示为基本遗传算法的流程图。

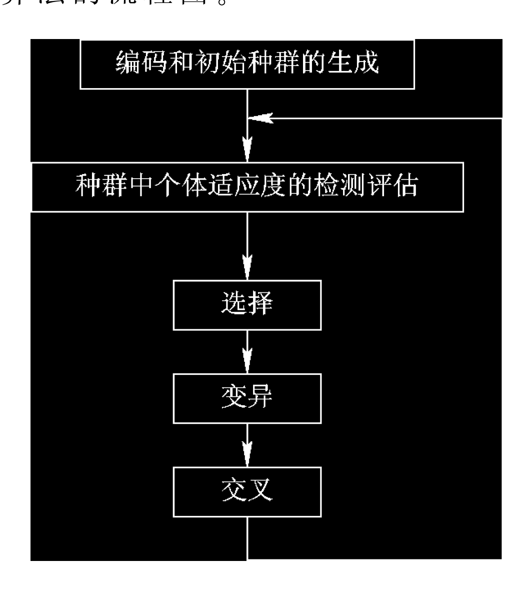



图2.3 遗传算法的基本流程图

### 2.2.2 基本遗传算法的步骤

**1. 染色体编码(Chromosome Coding)与解码(Decode)**

基本遗传算法使用固定长度的二进制符号串来表示群体中的个体，其等位基因由二值{0，1}所组成。初始群体中各个个体的基因可用均匀分布的随机数来生成。例如：

$X$＝100111001000101101 就可表示一个个体，该个体的染色体长度是 $n=18$。

（1）编码：设某一参数的取值范围为$[U_1, U_2]$，我们用长度为 $k$ 的二进制编码符号来表示该参数，则它总共产生 $2^k$ 种不同的编码，可使参数编码时的对应关系为：

$$000000\cdots0000=0 \longrightarrow U_1$$
$$000000\cdots0001=1 \longrightarrow U_1+\delta$$
$$000000\cdots0010=2 \longrightarrow U_1+2\delta$$
$$\vdots \qquad \vdots \qquad \vdots$$
$$111111\cdots1111=2^k-1 \longrightarrow U_2$$

其中，$\delta=\dfrac{U_2-U_1}{2^k-1}$。

（2）解码：假设某一个体的编码为 $b_k b_{k-1} b_{k-2} \cdots b_2 b_1$，则对应的解码公式为

$$X = U_1 + \left(\sum_{i=1}^{k} b_i \cdot 2^{i-1}\right) \cdot \frac{U_2-U_1}{2^k-1} \qquad (2.2)$$

例如：设有参数 $X \in [2, 4]$，现用 5 位二进制编码对 $X$ 进行编码，得 $2^5=32$ 个二进制串（染色体）：

00000，00001，00010，00011，00100，00101，00110，00111

01000，01001，01010，01011，01100，01101，01110，01111

10000，10001，10010，10011，10100，10101，10110，10111

11000，11001，11010，11011，11100，11101，11110，11111

对于任一个二进制串，只要代入公式(2.2)，就可得到对应的解码，如 $x_{22}=10101$，它对应的十进制为 $\sum_{i=1}^{5} b_i \cdot 2^{i-1}=1+0\times2+1\times2^2+0\times2^3+1\times2^4=21$，则对应参数 $X$ 的值为 $2+21\times\dfrac{4-2}{2^5-1}=3.3548$。

**2.** 个体适应度的检测评估

基本遗传算法按与个体适应度成正比的概率来决定当前群体中各个个体遗传到下一代群体中的机会多少。为了正确估计这个概率，要求所有个体的适应度必须为非负数。所以，根据不同种类的问题，需要预先确定好由目标函数值到个体适应度之间的转换规律，特别是要预先确定好当目标函数值为负数时的处理方法。例如，可选取一个适当大的正数 $c$，使个体的适应度为目标函数值加上正数 $c$。

**3.** 遗传算子

基本遗传算法使用下列三种遗传算子：

（1）选择运算使用比例选择算子。比例选择因子是利用比例于各个个体适应度的概率决定其子孙的遗留可能性。若设种群数为 $M$，个体 $i$ 的适应度为 $f_i$，则个体 $i$ 被选取的概率为

$$P_i = f_i \Big/ \sum_{k=1}^{M} f_k$$

当个体选择的概率给定后，产生$[0, 1]$之间的均匀随机数来决定哪个个体参加交配。若个体的选择概率大，则能被多次选中，它的遗传基因就会在种群中扩大；若个体的选择概率小，则被淘汰。

（2）交叉运算使用单点交叉算子。只有一个交叉点位置，任意挑选经过选择操作后种群中两个个体作为交叉对象，随机产生一个交叉点位置，两个个体在交叉点位置互换部分基因码，形成两个子个体，如图 2.4 所示。




图 2.4　单点交叉示意图

（3）变异运算使用基本位变异算子或均匀变异算子。为了避免问题过早收敛，对于二进制的基因码组成的个体种群，实现基因码的小概率翻转，即 0 变为 1，而 1 变为 0，如图 2.5 所示。




图 2.5　变异操作示意图

**4.** 基本遗传算法的运行参数

基本遗传算法有下列 4 个运行参数需要预先设定，即 $M$，$T$，$P_c$，$P_m$。

$M$ 为群体大小，即群体中所含个体的数量，一般取为 20～100；

$T$ 为遗传算法的终止进化代数，一般取为 100～500；

$P_c$ 为交叉概率，一般取为 0.4～0.99；

$P_m$ 为变异概率，一般取为 0.0001～0.1。

### 2.2.3　遗传算法的具体例证

下面通过具体的例子介绍遗传算法的实际工作过程。

假设目标函数为

$$\max f(x_1, x_2) = 21.5 + x_1 \sin(4\pi x_1) + x_2 \sin(20\pi x_2) \tag{2.3}$$

$$\left.\begin{aligned} -3.0 \leqslant x_1 \leqslant 12.1 \\ 4.1 \leqslant x_2 \leqslant 5.8 \end{aligned}\right\} \tag{2.4}$$

如图 2.6 所示，该函数有多个局部极值点。

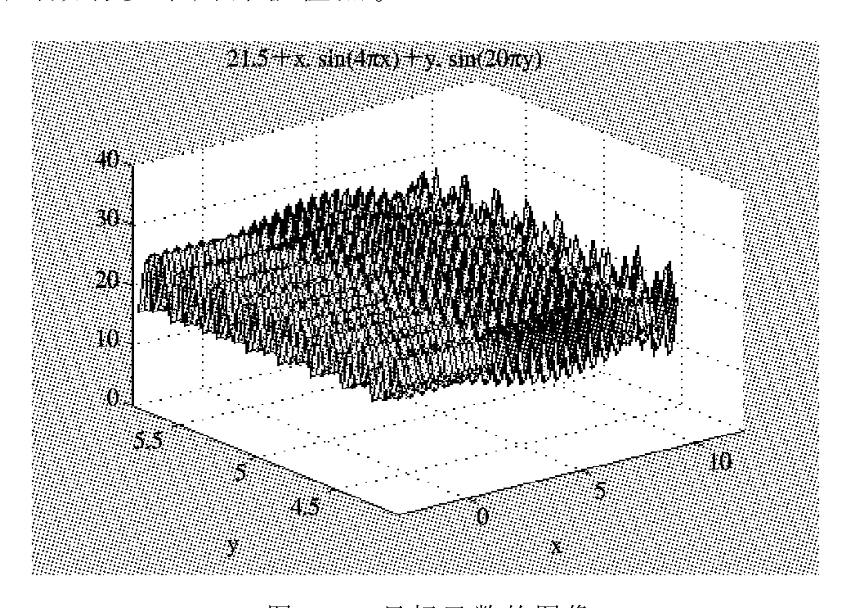



图 2.6　目标函数的图像

下面介绍求解该优化问题的遗传算法的构造过程。

第一步，确定决策变量和约束条件。

式(2.4)给出，决策变量为 $x_1$，$x_2$，约束条件为 $-3.0\leqslant x_1\leqslant 12.1$，$4.1\leqslant x_2\leqslant 5.8$。

第二步，建立优化模型。

式(2.3)已给出了问题的数学模型。

第三步，确定编码方法。

要进行编码工作，即将变量转换成二进制串。串的长度取决于所要求的精度。例如，变量 $x_1$ 的区间是 $[a_1, b_1]$，要求的精度是小数点后4位，也就意味着每个变量应该被分成至少 $(b_1-a_1)\times 10^4$ 个部分。对一个变量的二进制串位数(用 $m_j$ 表示)，用以下公式计算：

$$2^{m_j-1} < (b_j - a_j)\times 10^4 \leqslant 2^{m_j}-1$$

第四步，确定解码方法。

从二进制串返回一个实际的值可用下面的公式来实现：

$$x_j = a_j + decimal(substring_j)\times \frac{b_j - a_j}{2^{m_j}-1} \tag{2.5}$$

其中，$decimal(substring_j)$ 代表变量 $x_j$ 的十进位值。

不妨设要求的精度为小数点后4位，则目标函数的两个变量 $x_1$ 和 $x_2$ 可以转换为下面的串：

$$(12.1-(-3.0))\times 10\ 000 = 151\ 000$$

$$2^{17} < 151\ 000 \leqslant 2^{18}, \qquad m_1 = 18$$

$$(5.8-4.1)\times 10\ 000 = 17\ 000$$

$$2^{14} < 17\ 000 \leqslant 2^{15}, \qquad m_2 = 15$$

$$m = m_1 + m_2 = 18 + 15 = 33$$

这样，一个染色体串是33位，如图2.7所示。

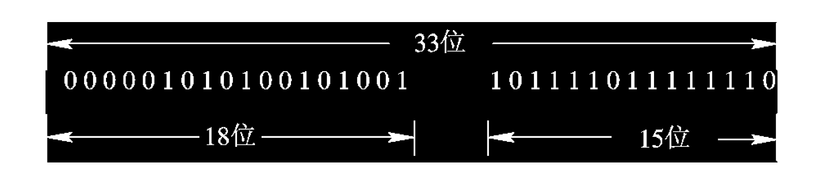



图2.7　一个染色体二进制串

对应的变量 $x_1$ 和 $x_2$ 的实值如表2.1所示。

表 **2.1**　染色体二进制与十进制比较

| 变量 | 二进制数 | 十进制数 |
| --- | --- | --- |
| $x_1$ | 000001010100101001 | 5417 |
| $x_2$ | 101111011111110 | 24 318 |

$$x_1 = -3.0 + 5417\times \frac{12.1-(-3.0)}{2^{18}-1} = -2.687\ 969$$

$$x_2 = 4.1 + 24\ 318\times \frac{5.8-4.1}{2^{15}-1} = 5.361\ 653$$

初始种群可随机生成如下：

$U_1=[000001010100101001101111011111110]$

$U_2=[001110101110011000000010101001000]$

$U_3=[111000111000001000010101001000110]$

$U_4=[100110110100101101000000010111001]$

$U_5=[000010111101100010001110001101000]$

$U_6=[111110101011011000000010110011001]$

$U_7=[110100010011111000100110011101101]$

$U_8=[001011010100001100010110011001100]$

$U_9=[111110001011101100011101000111101]$

$U_{10}=[111101001110101010000010101101010]$

相对应的十进制的实际值为

$U_1=[x_1, x_2]=[-2.687\ 969, 5.361\ 653]$

$U_2=[x_1, x_2]=[0.474\ 101, 4.170\ 144]$

$U_3=[x_1, x_2]=[10.419\ 457, 4.661\ 461]$

$U_4=[x_1, x_2]=[6.159\ 951, 4.109\ 598]$

$U_5=[x_1, x_2]=[-2.301\ 286, 4.477\ 282]$

$U_6=[x_1, x_2]=[11.788\ 084, 4.174\ 346]$

$U_7=[x_1, x_2]=[9.342\ 067, 5.121\ 702]$

$U_8=[x_1, x_2]=[-0.330\ 256, 4.694\ 977]$

$U_9=[x_1, x_2]=[11.671\ 267, 4.873\ 501]$

$U_{10}=[x_1, x_2]=[11.446\ 273, 4.171\ 908]$

第五步，确定个体评价方法。

对一个染色体串的适应度评价由下列三个步骤组成：

(1) 将染色体串进行反编码，转换成真实值。在本例中，意味着将二进制串转为实际值：

$$x^k=(x_1^k, x_2^k), \quad k=1, 2, \cdots \tag{2.6}$$

(2) 评价目标函数 $f(x^k)$。

(3) 将目标函数值转为适应度。对于极大值问题，适应度就等于目标函数值，即

$$eval(U_k)=f(x^k), \quad k=1, 2, \cdots \tag{2.7}$$

在遗传算法中，评价函数起着自然进化中环境的角色，它通过染色体的适应度对其进行评价。上述染色体的适应度值如下：

$eval(U_1)=f(-2.687\ 969, 5.361\ 653)=19.805\ 119$

$eval(U_2)=f(0.474\ 101, 4.170\ 144)=17.370\ 896$

$eval(U_3)=f(10.419\ 457, 4.661\ 461)=9.590\ 546$

$eval(U_4)=f(6.159\ 951, 4.109\ 598)=29.406\ 122$

$eval(U_5)=f(-2.301\ 286, 4.477\ 282)=15.686\ 091$

$eval(U_6)=f(11.788\ 084, 4.174\ 346)=11.900\ 541$

$eval(U_7)=f(9.342\ 067, 5.121\ 702)=17.958\ 717$

$eval(U_8) = f(-0.330\ 256, 4.694\ 977) = 19.763\ 190$

$eval(U_9) = f(11.671\ 267, 4.873\ 501) = 26.401\ 669$

$eval(U_{10}) = f(11.446\ 273, 4.171\ 908) = 10.252\ 480$

由以上数据可以看出，上述染色体中最健壮的是 $U_4$，最虚弱的是 $U_3$。

第六步，设计遗传算子和确定遗传算法的运行参数。

(1) 选择运算使用轮盘选择算子。

为基础的概率分配来选择新的种群。其步骤如下：

① 计算各染色体 $U_k$ 的适应度值 $eval(U_k)$：

$$eval(U_5) = f(x), \quad k = 1, 2, \cdots \tag{2.8}$$

② 计算群体的适应度值总和：

$$F = \sum_{k=1}^{pop-size} eval(U_k) \tag{2.9}$$

③ 计算对应于每个染色体 $U_k$ 的选择概率 $P_k$：

$$P_k = \frac{eval(U_k)}{F} \tag{2.10}$$

④ 计算每个染色体 $U_k$ 的累计概率 $Q_k$：

$$Q_k = \sum_{j=1}^{k} P_j, \quad j = 1, 2, \cdots \tag{2.11}$$

在实际工作中，选择新种群的一个染色体按以下步骤完成：

① 生成一个[0，1]间的随机数 $r$。

② 如果 $r \geqslant Q_1$，就选择染色体 $U_1$；否则，选择第 $k$ 个染色体 $U_k(2 \leqslant k \leqslant pop-size)$ 使得

$$Q_{k-1} \leqslant r \leqslant Q_k$$

那么，本例中种群的适应度总和为

$$F = \sum_{k=1}^{10} eval(U_k) = 178.135\ 372$$

对应于每个染色体 $U_k(k=1, 2, \cdots, 10)$的选择概率 $P_k$ 如下：

$P_1 = 0.111\ 180$ $P_2 = 0.097\ 515$ $P_3 = 0.053\ 839$ $P_4 = 0.165\ 077$ $P_5 = 0.088\ 057$

$P_6 = 0.066\ 806$ $P_7 = 0.100\ 815$ $P_8 = 0.110\ 945$ $P_9 = 0.148\ 211$ $P_{10} = 0.057\ 554$

对应于每个染色体 $U_k(k=1, 2, \cdots, 10)$的累计概率 $Q_k$ 如下：

$Q_1 = 0.111\ 180$ $Q_2 = 0.208\ 695$ $Q_3 = 0.262\ 534$ $Q_4 = 0.427\ 611$ $Q_5 = 0.515\ 668$

$Q_6 = 0.582\ 475$ $Q_7 = 0.683\ 290$ $Q_8 = 0.794\ 234$ $Q_9 = 0.942\ 446$ $Q_{10} = 1.000\ 000$

现在，我们转动轮盘 10 次，每次选择一个新种群中的染色体。假设这 10 次中生成的[0，1]间的随机数如下：

0.301 431　0.322 062　0.766 503　0.881 893　0.350 871

0.583 392　0.177 618　0.343 242　0.032 685　0.197 577

第一个随机数 $r_1 = 0.301\ 431$ 大于 $Q_3$，小于 $Q_4$，这样染色体 $U_4$ 被选中；第二个随机数 $r_2 = 0.322\ 062$ 也大于 $Q_3$，小于 $Q_4$，于是染色体 $U_4$ 被再次选中。最终，新的种群由下列染色体组成：

$U_1$ =[1 001101 10100 1 01101000000010 1 1 100 1]　　($U_4$)

$U_2$ =[1 001101 10100 1 01101000000010 1 1 100 1]　　($U_4$)

$U_3$ =[00101 101 010000110001011 001 10011 00]　　($U_8$)

$U_4$ =[11111000101 1 101100011 1010001 1 1 1101]　　($U_9$)

$U_5$ =[100110110100101101000000001 01 1 10 0 1]　　($U_4$)

$U_6$ =[1101000100111110 00 10011 00111 01101]　　($U_7$)

$U_7$ =[0011101011 100110000000 1010100 1000]　　($U_2$)

$U_8$ =[1001101101001011 010000 0001011 1001]　　($U_4$)

$U_9$ =[000001010 10010100110 11110111 11110]　　($U_1$)

$U_{10}$ =[0 011101 0111001100000000101010001000]　　($U_2$)

(2) 交叉运算使用单点交叉算子。

随机选择一个染色体串的节点，然后交换两个父辈节点右端部分来产生子辈。假设两个父辈染色体如下所示(节点随机选择在染色体串的第 17 位基因)：

↓

$U_1$ =[100110110100101101000000010111001]

$U_2$ =[001011010100001100010110011001100]

假设交叉概率为 $P_0$ =25%，即在平均水平上有 25%的染色体进行了交叉。交叉操作的过程如下：

开始

$k \leftarrow 0$；

当 $k \leqslant 10$ 时继续

$r_k \leftarrow$ [0，1]之间的随机数；

如果 $r_k < 0.25$，则

选择 $U_k$ 为交叉的一个父辈；

结束

$k \leftarrow k+1$；

结束

结束

假设随机数如下：

0.625 721　　0.266 823　　0.288 644　　0.295 114　　0.163 274

0.567 461　　0.085 940　　0.392 865　　0.770 714　　0.548 656

那么，就意味着染色体 $U_5$ 和 $U_7$ 被选中作为交叉的父辈。在这里，我们随机选择一个[1，32]间的整数(因为 33 是整个染色体串的长度)作为交叉点。假设生成的整数 *pos* 为 1，那么两个染色体从第一位分割，新的子辈在第一位右端的部分互换而生成，即

$U_5$ =[100110110100101101000000010111001]

$U_7$ =[001110101110011000000001010100 1000]

↓

$U_5^*$ =[101110101110011000000001010101000]

$U_7^*$ =[000110110100101101000000010111001]

（3）变异运算使用基本位变异算子。

假设染色体 $U_1$ 的第 18 位基因被选作变异，即如果该位基因是 1，则变异后就为 0。于是，染色体在变异后将是：

$$U_1=[100110110100101101000000010111001]$$
$$\downarrow$$
$$U_1^*=[100110110100101100000000010111001]$$

将变异概率设为 $P_m=0.01$，就是说，希望在平均水平上，种群内所有基因的 1% 要进行变异。在本例中，共有 33×10 ＝330 个基因，希望在每一代中有 3.3 个变异的基因，每个基因变异的概率是相等的。因此，我们要生成一个位于[0，1]间的随机数系列 $r_k(k=1, 2, \cdots, 330)$。假设表 2.2 中列出的基因将进行变异。

表 **2.2**　基因变异示例

| 基因位置 | 染色体位置 | 基因位数 | 随机数 |
|---|---|---|---|
| 105 | 4 | 6 | 0.009 857 |
| 164 | 5 | 32 | 0.003 113 |
| 199 | 7 | 1 | 0.000 946 |
| 329 | 10 | 32 | 0.001 282 |

在变异完成后，得到了最终的下一代种群：

$U_1^*=[100110110100101101000000010111001]$

$U_2^*=[100110110100101101000000010111001]$

$U_3^*=[001011010100001100010110011001100]$

$U_4^*=[111111001011101100011101000111101]$

$U_5^*=[101110101110011000000010101001010]$

$U_6^*=[110100010011111000100110011101101]$

$U_7^*=[100110110100101101000000010111001]$

$U_8^*=[100110110100101101000000010111001]$

$U_9^*=[000001010100101001101111011111110]$

$U_{10}^*=[001110101110011000000010101001010]$

相对应的变量$[x_1, x_2]$的十进制值和适应度值为

$f(6.159\ 951, 4.109\ 598)=29.406\ 122$

$f(6.159\ 951, 4.109\ 598)=29.406\ 122$

$f(-0.330\ 256, 4.694\ 977)=19.763\ 190$

$f(11.907\ 206, 4.873\ 501)=5.702\ 781$

$f(8.024\ 130, 4.170\ 248)=19.910\ 25$

$f(9.342\ 067, 5.117\ 02)=17.958\ 717$

$f(6.159\ 951, 4.109\ 598)=29.406\ 122$

$f(6.159\ 951, 4.109\ 598)=29.406\ 122$

$f(-2.687\ 969, 5.361\ 653)=19.805\ 119$

$f(0.474\ 101, 4.170\ 248)=17.370\ 896$

至此，完成了遗传算法第一代的流程。

设计终止代数为1000。在第491代，得到了最佳的染色体：

$U^* =[11111000000011\ 1000111101001010110]$

个体随着进化过程的进行，群体中适应度较低的一些个体被逐渐淘汰，而适应度较高的一些个体会越来越多，并且更加集中在$U^*$附近，最终就可搜索到问题的最优点$U^*$。$U^*$对应的十进制为$x_1^*=11.631\ 407$，$x_2^*=5.724\ 824$，得适应度值为

$$eval(U^*)=f(11.631\ 407, 5.724\ 824)=38.818\ 208$$

即目标函数的最大值为$f(x_1^*, x_2^*)=38.818\ 208$。

## 2.3 改进的遗传算法

尽管遗传算法有许多优点，也有许多专家学者对遗传算法进行不断研究，但目前存在的问题依然很多，如：

(1) 适应度值标定方式多种多样，没有一个简洁、通用的方法，不利于对遗传算法的使用。

(2) 遗传算法的早熟现象(即很快收敛到局部最优解而不是全局最优解)是迄今为止最难处理的关键问题。

(3) 快要接近最优解时在最优解附近左右摆动，收敛较慢。

本节根据遗传算法所存在的这些问题分别从适应度值函数标定和增加群体多样性两方面着手解决。

遗传算法通常需要解决以下问题，如确定编码方案，适应度函数标定，选择遗传操作方式及相关控制参数，停止准则确定等。相应地，为改进简单遗传算法的实际计算性能，很多学者的改进工作也是分别从参数编码、初始群体设定、适应度函数标定、遗传操作算子、控制参数的选择以及遗传算法的结构等方面提出的。自从1975年J. H. Holland系统提出遗传算法的完整结构和理论以来，众多学者一直致力于推动遗传算法的发展，对编码方式、控制参数的确定和交叉机理等进行了深入的研究，提出了各种变形的遗传算法。其基本途径概括起来主要有下面几个方面：

(1) 改进遗传算法的组成成分或使用技术，如选用优化控制参数、适合问题特性的编码技术等。

(2) 采用混合遗传算法(Hybrid Genetic Algorithm)。

(3) 采用动态自适应技术，在进化过程中调整算法控制参数和编码精度。

(4) 采用非标准的遗传操作算子。

(5) 采用并行算法。

在许多资料中都介绍了七种改进遗传算法：

（1）分层遗传算法(Hierarchic Genetic Algorithm)；

（2）CHC 算法；

（3）Messy 遗传算法；

（4）自适应遗传算法(Adaptive Genetic Algorithm)；

（5）基于小生境技术的遗传算法(Niched Genetic Algorithm，简称 NGA)。

（6）并行遗传算法(Parallel Genetic Algorithm)；

（7）混合遗传算法：

① 遗传算法与最速下降法相结合的混合遗传算法；

② 遗传算法与模拟退火法(Simulated Annealing)相结合的混合遗传算法。

下面介绍几种改进的遗传算法。

## 2.3.1 改进的遗传算法一

在改进的遗传算法中，改进的三个算子通常是 GA 算法中的交叉操作，是随机取两个染色体进行单点交叉操作(也可用其他的交叉操作，如多点交叉、树交叉、部分匹配交叉等)，即在以高适应度模式为祖先的“家族”中取一点，但这种取法有其片面性。经证明，简单遗传算法在任何情况下(交叉概率 $P_c$，变异概率 $P_m$，任意初始化，任意交叉算子，任意适应度函数)都是不收敛的，即不能搜索到全局最优解；而通过改进的遗传算法，即在选择作用前(或后)保留当前最优解，则能保证收敛到全局最优解。尽管人们证明了改进的遗传算法最终能收敛到最优解，但收敛到最优解所需的时间可能是很长的。另外，早熟问题是遗传算法中不可忽视的现象，其具体表现为：

（1）群体中所有的个体都陷于同一极值而停止进化。

（2）接近最优解的个体总是被淘汰，进化过程不收敛。

对此可以采用以下方法来解决：

（1）动态确定变异概率，既可防止优良基因因为变异而遭破坏，又可在陷于局优解时为种群引入新的基因。

（2）改进选择方式，放弃赌轮选择，以避免早期的高适应度个体迅速占据种群和后期的种群中因个体的适应度相差不大而导致种群停止进化；赌轮选择方式就会使每一个个体都获得复制一份的机会，体现不出好的个体的竞争力，无法实现遗传算法的优胜劣汰的原则。鉴于此，这里用一种基于种群的按个体适应度大小排序的选择算法来代替赌轮选择方法。其过程描述如下：

```
first( )    {将种群中的个体按适应度大小进行排序；}
while        种群还没有扫描完
do          {排在前面的个体复制两份；中间的复制一份；后面的不复制；}
```

择优交叉在解决过早收敛问题时，通常习惯于采用限制优良个体的竞争力(高适应度个体的复制份数)的方法。这样无疑会降低算法的进化速度，增大算法的时间复杂度，降低算法的性能。由于种群的基因多样性可以减小陷入局优解的可能，而加快种群进化速度又可以提高算法的整体性能。为了解决这一对矛盾，尝试一种在不破坏种群的基因多样性的前提下加快种群的进化速度的方法，这一方法描述如下：在随机选择出父本和母本以后，按照交叉方法(单点交叉、多点交叉和一致交叉)进行 $n$ 次交叉，产生 $2n$ 个个体，再从这 $2n$

个个体中挑选出最优的两个个体加入新的种群中。这样既保存了父本和母本的基因，又在进化的过程中大大地提高了种群中个体的平均性能。

基于以上的分析，改进的遗传算法一描述如下：

（1）在初始种群中，对所有的个体按其适应度大小进行排序，然后计算个体的支持度和置信度；

（2）按一定的比例复制（即将当前种群中适应度最高的两个个体结构完整地复制到待配种群中）；

（3）按个体所处的位置确定其变异概率并变异；按优良个体复制 4 份，劣质个体不复制的原则复制个体；

（4）从复制组中随机选择两个个体，对这两个个体进行多次交叉，从所得的结果中选择一个最优个体存入新种群；

（5）若满足结束条件，则停止，不然，跳转第(1)步，直至找到所有符合条件的规则。

该算法的优点是在各代的每一次演化过程中，子代总是保留了父代中最好的个体，以在“高适应度模式为祖先的家族方向”搜索出更好的样本，从而保证最终可以搜索到全局最优解。

### 2.3.2 改进的遗传算法二

改进的遗传算法二的步骤如下：

（1）划分寻优空间。字符串中表示各个变量 $x_i$ 的子字符串的最高位 $B_i^{n_i}$（最左边）可以是 0 或 1（用 $b$ 表示，下同）。据此存在一种划分，可以把字符串划分成对等的两个子空间。假设有 $m$ 个变量，则存在 $m$ 个这种划分方式，可以形成 $m$ 对子空间，用集合表示为

$$A_i^b = \{S \mid B_i^{n_i} = b\}, \quad i = 1, 2, \cdots, m; b = 0, 1$$

为划分区间，在此将个体依适应度值降序排列为

$$S_1', S_2', S_3', \cdots, S_{np}', \quad f(\varphi(S_i')) \geqslant f(\varphi(S_{i-1}')), i = 1, 2, 3, \cdots, np$$

（2）设计空间退化。在演化到某一代时，如果适应值最高的前 $np_0$ 个（$np_0$ 取群体规模的一个事先确定的比例，这里取 $0.3np$）个体都位于同一字符串子空间（如 $A_k^b$）内：

$$S_j' \in A_k^b, \quad i = 1, 2, \cdots, np_0; b = 0 \text{ 或 } 1$$

可以认为最优点以很大概率落入 $A_k^b$ 中，以此作为下一代的寻优空间。对应变量为

$$\begin{cases} x_i \in \begin{cases} [x_i^L, x_i^U] & i \neq k \\ [x_i'^L, x_i'^U] & i = k \end{cases} \\ [x_i'^L, x_i'^U] = \begin{cases} [x_k^L, x_k^U] & b = 0 \\ [x_k'^L, x_k'^U] & b = 1 \end{cases}, x_k^m = \dfrac{1}{2}(x_k^L + x_k^U) \end{cases} \tag{2.12}$$

由于在该空间内表示第 $i$ 个变量的子字符串 $S_i' = B_i^{n_i} B_i^{n_i-1} \cdots B_i^1$ 中最高位和标准遗传算法中的最高位一样，为提高编码效率，提高变量表达精度，同时保证各基因位按模式定理解释时含义不变，将 $S_i'$ 中的各位基因位从左边第二位 $B_i^{n_i-1}$ 开始，依次左移一位：

$$B_i^{j+1} \Leftarrow B_i^j, \quad j = n_i - 1, n_i - 2, \cdots, 1$$

而最后一位由随机数填充。为了保护最优个体，使其在区间退化时对应变量不变，最优个

体的最后位与移动前的首位一致。由于设计空间的不断退化，每个变量的串长 $n_i$ 无需太长，取 4～6 位即可，不影响精度。

(3) 寻优空间的移动。如果当前最优解的某个分量 $x_k$ 处在当前设计空间的边界，该变量对应的子串的各位相同，均为 0 或 1，则认为最优解有可能在当前寻优区间以外。此时，在该分量方向移动寻优空间，以避免寻优空间缩减而导致失去最优解。可以取移动距离为 $2d_k$，$d_k$ 为沿 $x_k$ 方向相邻两个离散点间的距离：

$$d_k = \frac{x_k^U - x_k^L}{2^{n_k} - 1} \tag{2.13}$$

移动方法是调整边界：

$$[x_i^L, x_i^U] = \begin{cases} [x_k^L - 2d_k, x_k^U - 2d_k] & b = 0 \\ [x_k^L + 2d_k, x_k^U + 2d_k] & b = 1 \end{cases} \tag{2.14}$$

然后改变对应子串，改变方法是把该子串作为二进制数，当 $b=1$ 时减 2，反之加 2。这样操作保证了处在移动前后两个空间的重叠部分的个体处在设计空间的同一位置上。当有进位或借位发生时，说明该点将被移出当前寻优空间，略去进位或借位，就会落入新移入的那部分寻优空间内，可以理解为随机产生的新个体。

### 2.3.3 改进的遗传算法三

标准遗传算法是具有“生成＋检测”的迭代过程的搜索算法。遗传算法采用一种群体搜索策略和群体中个体之间的信息交换、搜索，不依赖于梯度信息。但标准遗传算法存在一些不足，下面是标准遗传算法中存在的主要问题及解决方案。

对早熟收敛和后期搜索迟钝的解决方案：有条件的最佳保留机制；采用遗传—灾变算法；采用适应度比例机制和个体浓度选择机制的加权和；引入主群和属群的概念；适应度函数动态定标；多种群并行进化及自适应调整控制参数相结合的自适应并行遗传算法，对重要参数的选择采用自适应变化而非固定不变。

采用的具体方法如下：

(1) 交叉和变异算子的改进和协调采用。

① 将进化过程划分为渐进和突变两个不同阶段；

② 采用动态变异；

③ 运用正交设计或均匀设计方法设计新的交叉和变异算子。

(2) 采用与局部搜索算法相结合的混合遗传算法，解决局部搜索能力差的问题。

(3) 采用有条件的替代父代的方法，解决单一的群体更新方式难以兼顾多样性和收敛性的问题。

(4) 收敛速度慢的解决方法：

① 产生好的初始群体；

② 利用小生境技术；

③ 使用移民技术；

④ 采用自适应算子；

⑤ 采用与局部搜索算法相结合的混合遗传算法；

⑥ 对算法的参数编码采用动态模糊控制；

⑦ 进行未成熟收敛判断。

遗传算法中包含如下5个基本要素：参数编码、初始群体的设定、适应度函数的设计、操作设计和控制参数设定。这5个要素构成遗传算法的核心内容。接下来将从初始群体产生、选择算子的改进、遗传算法重要参数的选择、群体更新方式、适应度函数的选取等几个方面对标准遗传算法进行改进。

**1．** 初始群体的产生

初始群体的特性对计算结果和计算效率均有重要影响。要实现全局最优解，初始群体在解空间中应尽量分散。标准遗传算法是按预定或随机方法产生一组初始解群体，这样可能导致初始解群体在解空间分布不均匀，从而影响算法的性能。要得到一个好的初始群体，可以将一些实验设计方法，如均匀设计或正交设计与遗传算法相结合。其原理为：首先根据所给出的问题构造均匀数组或正交数组，然后执行如下算法产生初始群体：

(1) 将解空间划分为 $S$ 个子空间；

(2) 量化每个子空间，运用均匀数组或正交数组选择 $M$ 个染色体；

(3) 从 $M\times S$ 个染色体中，选择适应度函数最大的 $N$ 个作为初始群体。

这样可保证初始群体在解空间均匀分布。

另外，初始群体的各个个体之间应保持一定的距离，并定义相同长度的以某一常数为基的两个字符串中对应位不同的数量为两者间的广义海明距离。要求入选群体的所有个体之间的广义海明距离必须大于或等于某个设定值。初始群体采用这种方法产生能保证随机产生的各个个体间有较明显的差别，使它们能均匀分布在解空间中，从而增加获取全局最优解的可能。

**2．** 选择算子的改进

在标准遗传算法中，常根据个体的适应度大小采用“赌轮选择”策略。该策略虽然简单，但容易引起“早熟收敛”和“搜索迟钝”问题。有效的解决方法是采用有条件的最佳保留策略，即有条件地将最佳个体直接传递到下一代或至少等同于前一代，这样能有效防止“早熟收敛”。

也可以使用遗传—灾变算法，即在遗传算法的基础上，模拟自然界的灾变现象，提高遗传算法的性能。当判断连续数代最佳染色体没有任何进化，或者各个染色体已过于近似时，即可实施灾变。灾变的方法很多，可以突然增大变异概率或对不同个体实施不同规模的突变，以产生不同数目的大量后代等。用灾变的方法可以打破原有基因的垄断优势，增加基因的多样性，创造有生命力的新个体。

**3．** 遗传算法重要参数的选择

遗传算法中需要选择的参数主要有：染色体长度 $l$、群体规模 $n$、交叉概率 $P_c$ 和变异概率 $P_m$ 等，这些参数对遗传算法的性能影响也很大。染色体长度的选择对二进制编码来说取决于特定问题的精度，存在定长和变长两种方式。群体规模通常取20～200。一般来说，求解问题的非线性越大，$n$ 选择就应该越大。交叉操作和变异操作是遗传算法中两个起重要作用的算子。通过交叉和变异，一对相互配合又相互竞争的算子使其搜索能力得到飞速提高。交叉操作的作用是组合交叉两个个体中有价值的信息产生新的后代，它在群体进化期间大大加快了搜索速度；变异操作的作用是保持群体中基因的多样性，偶然、次要的(交叉率取很小)起辅助作用。在遗传算法的计算过程中，根据个体的具体情况，自适应

地改变 $P_c$、$P_m$ 的大小，将进化过程分为渐进和突变两个不同阶段：渐进阶段强交叉，弱变异，强化优势型选择算子；突变阶段弱交叉，强变异，弱化优势型选择算子。这样对提高算法的计算速度和效率是有利的。

自适应参数调整方案如下：

$$\delta = f_{max} - \bar{f} \tag{2.15}$$

式中，$f_{max}$ 为某代中最优个体适应度，$\bar{f}$ 为此代平均适应度。

**4**．适应度函数的设计

遗传算法中采用适应度函数值来评估个体性能并指导搜索，基本不用搜索空间的知识，因此，适应度函数的选取相当重要。性能不良的适应度函数往往会导致“骗”问题。适应度函数的选取标准是：规范性（单值、连续、严格单调）、合理性（计算量小）、通用性。Vasilies Retridis 提出在解约束优化问题时采用变化的适应度函数的方案。将问题的约束以动态方式合并到适应度函数中，即形成一个具有变化的惩罚项的适应度函数，用来指导遗传搜索。在那些具有许多约束条件而导致产生一个复杂搜索超平面的问题中，该方案能明显地以较大的概率找到全局最优解。

**5**．进化过程中动态调整子代个体

遗传算法要求在进行过程中保持群体规模不变。但为了防止早熟收敛，在进化过程可对群体中的个体进行调整，包括引入移民算子、过滤相似个体、动态补充子代新个体等。

移民算子是避免早熟的一种好方法。在移民的过程中不仅可以加速淘汰差的个体，而且可以增加解的多样性。所谓的移民机制，就是在每一代进化过程中以一定的淘汰率（一般取 15％～20％）将最差个体淘汰，然后用产生的新个体代替。

为了加快收敛速度，可采用滤除相似个体的操作，减少基因的单一性。删除相似个体的过滤操作为：对子代个体按适应度排序，依次计算适应度差值小于门限 $delta$ 的相似个体间的广义海明距离（相同长度的以 $a$ 为基的两个字符串中对应位不相同的数量称为两者间的广义海明距离）。如果同时满足适应度差值小于门限 $delta$，广义海明距离小于门限 $d$，就滤除其中适应度较小的个体。$delta$、$d$ 应适当选取，以提高群体的多样性。过滤操作后，需要引入新个体。从实验测试中发现，如果采用直接随机生成的方式产生新个体，适应度值都太低，而且对算法的全局搜索性能增加并不显著（例如，对于复杂的多峰函数很难跳出局部最优点）。因此，可使用从优秀的父代个体中变异产生的方法。该方法将父代中适应度较高的 $m$ 个个体随机进行若干次变异，产生出新个体，加入子代对个体。这些新个体继承了父代较优个体的模式片断，并产生新的模式，易于与其他个体结合生成新的较优子代个体。而且增加的新个体的个数与过滤操作删除的数量有关。如果群体基因单一性增加，则被滤除的相似个体数目增加，补充的新个体数目随之增加；反之，则只少量滤除相似个体，甚至不滤除，补充的新个体数目也随之减少。这样，就能动态解决群体由于缺乏多样性而陷入局部解的问题。

**6**．小范围竞争择优的交叉、变异操作

从加快收敛速度、全局搜索性能两方面考虑，受自然界中家庭内兄弟间竞争现象的启发，加入小范围竞争、择优操作。其方法是，将某一对父母 A、B 进行 $n$ 次（3～5 次）交叉、变异操作，生成 $2n$ 个不同的个体，选出其中一个最高适应度的个体，送入子代对个体中。反复随机选择父母对，直到生成设定个数的子代个体为止。这种方法实质是在相同父母的

情况下，预先加入兄弟间的小范围的竞争择优机制。另一方面，在标准遗传算法中，一对父母 X、Y 经遗传算法操作后产生一对子代个体 $xy_1$、$xy_2$、$x_1y$、$x_2y$，随后都被放入子代对个体，当进行新一轮遗传操作时，$xy_1$、$x_1y$ 可能作为新的父母对进行交叉配对，即“近亲繁殖”。而加入小范围竞争择优的交叉、变异操作，减少了在下一代中出现这一问题的几率。

## 2.3.4 改进的遗传算法四

改进的遗传算法四是指从适应度值标定和群体多样化两方面考虑，提出改进的遗传算法。

**1.** 适应度值标定

初始群体中可能存在特殊个体的适应度值超常（如很大）。为了防止其统治整个群体并误导群体的发展方向而使算法收敛于局部最优解，需限制其繁殖。在计算临近结束，遗传算法逐渐收敛时，由于群体中个体适应度值比较接近，继续优化选择较为困难，造成在最优解附近左右摇摆。此时应将个体适应度值加以放大，以提高选择能力，这就是适应度值的标定。针对适应度值标定问题提出以下计算公式：

$$f' = \frac{1}{f_{max} + f_{min} + \delta}(f + |f_{min}|) \tag{2.16}$$

式中，$f'$为标定后的适应度值，$f$ 为原适应度值，$f_{max}$为适应度函数值的一个上界，$f_{min}$为适应度函数值的一个下界，$\delta$ 为开区间(0，1)内的一个正实数。

若 $f_{max}$未知，可用当前代或目前为止的群体中的最大值来代替。若 $f_{min}$未知，可用当前代或目前为止群体中的最小值来代替。取 $\delta$ 的目的是防止分母为零和增加遗传算法的随机性。$|f_{min}|$是为了保证标定后的适应度值不出现负数。

由图 2.8 可见，若 $f_{max}$与 $f_{min}$差值越大，则角度 $\alpha$ 越小，即标定后的适应度值变化范围小，防止超常个体统治整个群体；反之则越大，标定后的适应度值变化范围增大，拉开群体中个体之间的差距，避免算法在最优解附近摆动现象发生。这样就可以根据群体适应度值放大或缩小，变更选择压力。

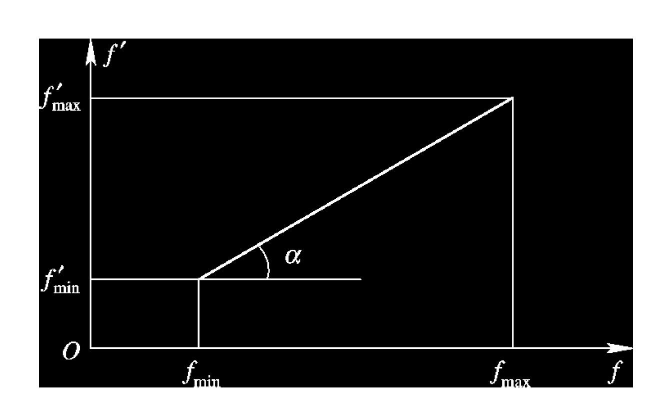



图 2.8　适应度值标定

**2.** 群体多样化

遗传算法在求解具有多个极值点的函数时，存在一个致命的弱点——早熟，即收敛到局部最优解而非全局最优解，这也是遗传算法最难解决的一个问题。遗传算法的早熟原因是交叉算子在搜索过程中存在着严重的成熟化效应。在起搜索作用的同时，不可避免的是群体多样化逐渐趋于零，从而逐渐减少了搜索范围，引起过早收敛。

为了解决这一问题，人们研究出很多方法：元算法、自适应遗传算法(AGA)、改进的自适应遗传算法(MAGA)等。可见，避免遗传算法早熟的关键是使群体呈多样化发展，也就是应使搜索点分布在各极值点所在的区域，如图2.9所示的$x_i$。

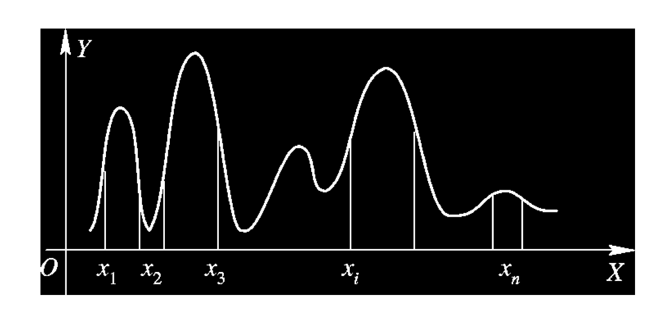



图2.9　多极值函数

简单遗传算法在进行优化计算时不是全局收敛。只有保证最优个体复制到下一代，才能保证其收敛性，也就是说，尽管遗传算法的基本作用对象是多个可行解且隐并行操作，仍然需要对其进行适当改进。为了增加群体的多样性，有效地避免早熟现象发生，引入了相似度的概念。

定义**2.1**　在遗传算法进行选择运算前，对群体中每两个个体逐位比较。如果两个个体中在相对应的位置上存在着相同的字符(基因)，则将相同字符数量定义为相似度$R$。

设置值$T$=适应度平均值，在群体中取大于$T$的个体进行个体相似程度判断。相似度低则表示这两个个体相似性差，当相似度值$R$超过个体长度$L/2$时，即认为这两个个体相似，如1011001和1101001的相似度值$R=5$，$L=7$，$R>L/2$，所以可以认为这两个个体具有相似性。相似性的判断实际上是确定群体中个体是否含有相同模式。剔除相似个体，选择不同模式的个体组成新的群体，可以增加群体的多样性，尤其是在计算初期，经过相似性判断后，能够有效避免早熟问题的产生。由此得出的改进遗传算法的步骤如下：

(1)个体按适应度值大小排序。

(2)求平均适应度值，以此作为阈值，选择适应度值大于平均适应度值的个体。

(3)判断相似程度，以最高适应度值为模板，去除相似个体。

(4)重复(3)，逐次以适应度值高的个体为模板，选择不同模板的个体组成群体。

(5)判断是否达到群体规模。如果是，则进行下一步交叉、变异等遗传操作；否则重复(4)。如果不能得到足够的群体规模，则去除的个体按适应度值大小顺序顺次补足群体所缺数量。

(6)判断是否满足结束要求。如果是，则结束，否则转到(1)。

为了避免过早陷入局部最优解，必须拓宽搜索空间，增加群体多样性。取平均适应度值作为阈值并以高于阈值的个体作为模板进行选择，有效鼓励高适应度值个体的竞争力。这样的处理，主要是为了增加群体的多样性和高适应度值的个体的主导地位，避免统一模式统治群体，从而误导搜索方向。当接近最优解时，由上面的运算步骤可以尽快收敛到最优解。这样既不增加群体规模，避免运算时间过长，还能保证收敛到全局最优解。

图2.10所示为改进遗传算法的程序流程图。

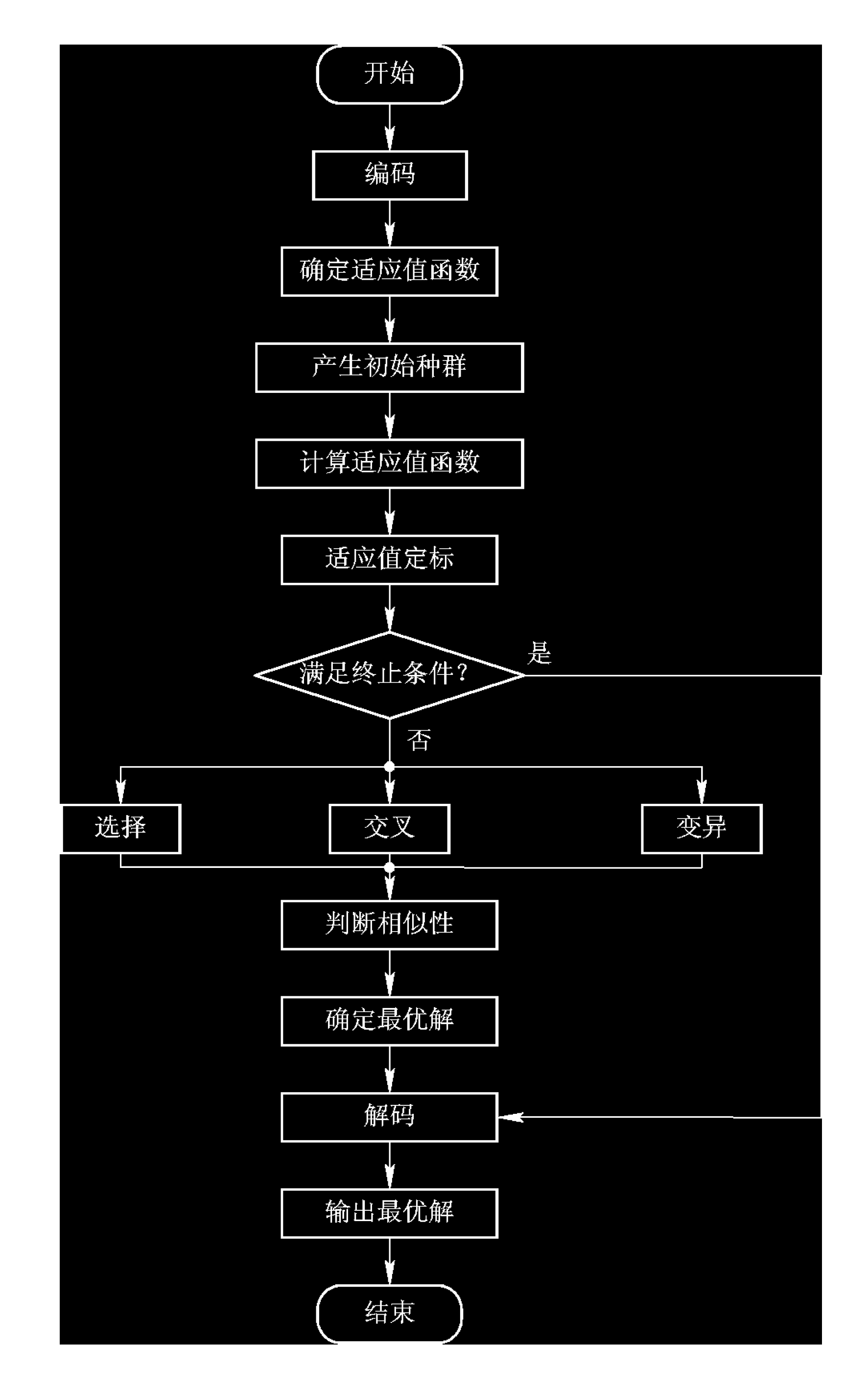



图 2.10　改进遗传算法的程序流程图

## 2.4　多目标优化中的遗传算法

前面讨论的都是单目标在给定区域上的最优化问题，而工程中经常遇到在多准则或多目标下设计和决策的问题，如果这些目标的改善是相互抵触的，则需要找到满足这些目标的最佳设计方案。利用遗传算法可以解决多目标优化问题。

### 2.4.1　多目标优化的概念

解决含多目标和多约束的优化问题称为多目标优化(Multi-objective Opimization)问题。在实际应用中，工程优化问题大多数是多目标优化问题，有时需要使多个目标在给定区域上都可能地达到最优的问题，目标之间一般都是互相冲突的。例如投资问题，一般我们都是希望所投入的资金量最少，风险最小，并且所获得的收益最大。这种多于一个的数

值目标的最优化问题就是多目标优化问题。多目标优化问题一般的数学模型可描述为

$$\begin{cases} V\text{-}\min & f(x) = [f_1(x), f_2(x), \cdots, f_n(x)]^{\mathrm{T}} \\ \text{s.t.} & x \in X \\ & X \subseteq R^m \end{cases} \tag{2.17}$$

式中，$V$-min 表示向量极小化，即向量目标函数 $f(x)=[f_1(x), f_2(x), \cdots, f_n(x)]^{\mathrm{T}}$ 中的各个子目标函数都尽可能地达到极小化。

下面先介绍多目标优化中最优解和 Pareto 最优解(Pareto Optimal Solution)的定义。

**定义 2.2** 设 $X \subseteq R^m$ 是多目标优化模型的约束集，$f(x) \in R^n$ 是多目标优化时的向量目标函数，有 $x_1 \in X$，$x_2 \in X$。若

$$f_k(x_1) \leqslant f_k(x_2), \quad \forall k = 1, 2, \cdots, n \tag{2.18}$$

并且

$$f_k(x_1) \leqslant f_k(x_2), \quad \exists k = 1, 2, \cdots, n \tag{2.19}$$

则称解 $x_1$ 比解 $x_2$ 优越。

**定义 2.3** 设 $X \subseteq R^m$ 是多目标优化模型的约束集，$f(x) \in R^n$ 是多目标优化时的向量目标函数，若有解 $x_1 \in X$，并且 $x_1$ 比 $X$ 中的所有其他解都优越，则称解 $x_1$ 是多目标优化模型的最优解。

由定义 2.3 可知，解 $x_1$ 使得所有的 $f(x_i)(i=1, 2, \cdots, n)$都达到最优(如图 2.11 所示)。但实际应用中一般不存在这样的解。

**定义 2.4** 设 $X \subseteq R^m$ 是多目标优化模型的约束集，$f(x) \in R^n$ 是多目标优化时的向量目标函数，若有解 $x_1 \in X$，并且不存在比 $x_1$ 更优越的解 $x$，则称 $x_1$ 是多目标最优化模型的 Pareto 最优解。

由定义 2.4 可知，多目标优化问题的 Pareto 最优解只是问题的一个可以接受的“非劣解”，并且一般多目标优化实际问题都存在多个 Pareto 最优解(如图 2.12 所示)。

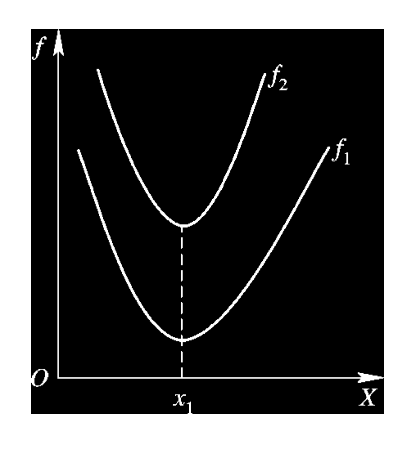



图 2.11 多目标优化问题的最优解

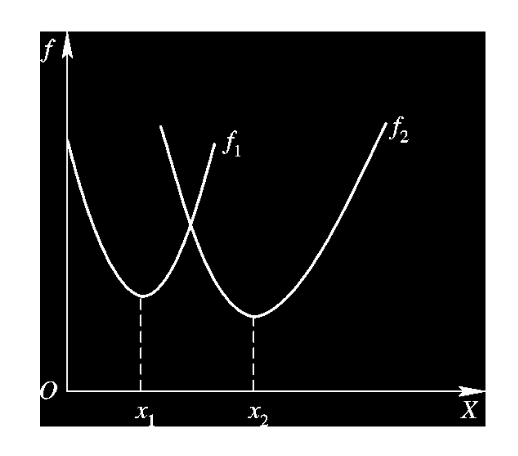



图 2.12 多目标优化问题的 Pareto 最优解

## 2.4.2 多目标优化问题的遗传算法

对于求解多目标优化问题的 Pareto 最优解，目前已有多种基于遗传算法的求解方法。下面介绍五种常用的方法。

**1．权重系数变换法**

对于一个多目标优化问题，若给其每个子目标函数 $f(x_i)(i=1, 2, \cdots, n)$ 赋予权重 $w_i(i=1, 2, \cdots, n)$，其中 $w_i$ 为相应的 $f(x_i)$ 在多目标优化问题中的重要程度，则各个子目标函数 $f(x_i)$ 的线性加权和表示为

$$u = \sum_{i=1}^{n} w_i \cdot f_i(x) \tag{2.20}$$

若将 $u$ 作为多目标优化问题的评价函数，则多目标优化问题就可转化为单目标优化问题，即可以利用单目标优化的遗传算法求解多目标优化问题。

**2．并列选择法**

并列选择法的基本思想是，先将群体中的全部个体按子目标函数的数目均等地划分为一些子群体，对每个子群体分配一个子目标函数，各个子目标函数在相应的子群体中独立地进行选择运算，各自选择出一些适应度高的个体组成一个新的子群体，然后再将所有这些新生成的子群体合并成一个完整的群体，在这个群体中进行交叉和变异运算，从而生成下一代的完整群体，如此不断地进行“分割—并列选择—合并”操作，最终可求出多目标优化问题的 Pareto 最优解。

图 2.13 所示为多目标优化问题的并列选择法的示意图。

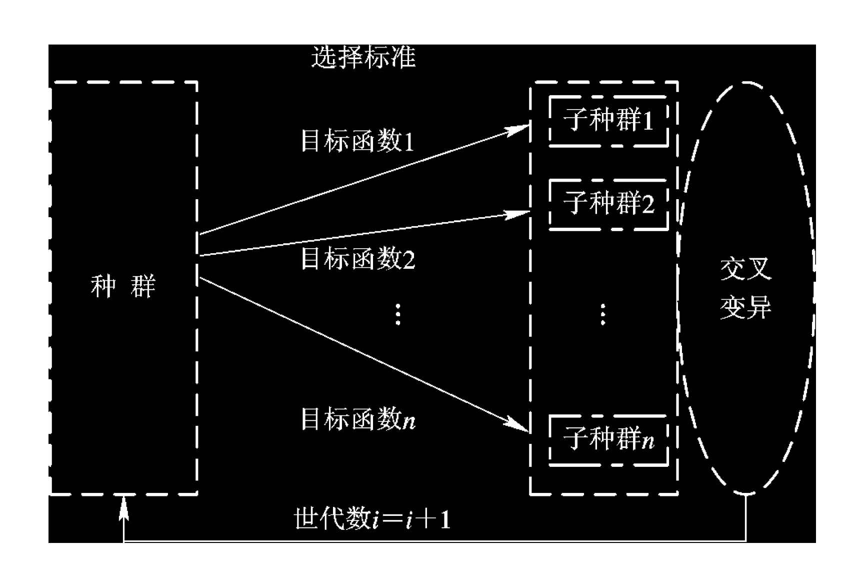



图 2.13　并列选择法的示意图

**3．排列选择法**

排列选择法的基本思想是，基于 Pareto 最优个体(Pareto 最优个体是指群体中的这样一个或一些个体，群体中的其他个体都不比它或它们更优越)，对群体中的各个个体进行排序，依据这个排列次序来进行进化过程中的选择运算，从而使得排在前面的 Pareto 最优个体将有更多的机会遗传到下一代群体中。如此这样经过一定代数的循环之后，最终就可求出多目标最优化问题的 Pareto 最优解。

**4．共享函数法**

求解多目标最优化问题时，一般希望所得到的解能够尽可能地分散在整个 Pareto 最优解集合内，而不是集中在其 Pareto 最优解集合内的某一个较小的区域上。为达到这个要求，可以利用小生境遗传算法的技术来求解多目标最优化问题，这种方法称为共享函数(Sharing Function)法，它将共享函数的概念引入到求解多目标最优化问题的遗传算法中。

算法对相同个体或类似个体的数量加以限制，以便能够产生出种类较多的不同的最优解。对于一个个体 $X$，在它的附近还存在有多少种、多大程度相似的个体，是可以度量的，这种度量值称为小生境数。小生境数的计算方法定义为

$$m_X = \sum_{Y \leqslant n} s[d(X, Y)] \tag{2.21}$$

式中，$s(d)$为共享函数，它是个体之间距离 $d$ 的单调递减函数。$d(X, Y)$可以定义为个体 $X$，$Y$ 之间的海明距离。

在计算出各个个体的小生境数之后，可以使小生境数较小的个体能够有更多的机会被选中，遗传到下一代群体中，即相似程度较小的个体能够有更多的机会被遗传到下一代群体中，这样就增加了群体的多样性，也增加了解的多样性。

**5.** 混合法

混合法的基本思想是，选择算子的主体使用并列选择法，然后通过引入保留最佳个体和共享函数的思想来弥补只使用并列选择法的不足之处。算法的主要过程为：

(1) 并列选择过程。按所求多目标优化问题的子目标函数的个数，将整个群体均等地划分成一些子群体，各个子目标函数在相应的子群体中产生其下一代子群体。

(2) 保留 Pareto 最优个体过程。对于子群体中的 Pareto 最优个体，不让其参与个体的交叉运算和变异运算，而是将这个或这些 Pareto 最优个体直接保留到下一代子群体中。

(3) 共享函数处理过程。若所得到的 Pareto 最优个体的数量已超过规定的群体规模，则需要利用共享函数的处理方法来对这些 Pareto 最优个体进行挑选，以形成规定规模的新一代群体。

# 第三章　遗传算法的理论基础

遗传算法有效性的理论依据为模式定理和积木块假设。模式定理保证了较优的模式(遗传算法的较优解)的样本呈指数级增长，从而满足了寻找最优解的必要条件，即遗传算法存在着寻找到全局最优解的可能性。而积木块假设指出，遗传算法具备寻找到全局最优解的能力，即具有低阶、短距、高平均适应度的模式(积木块)在遗传算子作用下，相互结合，能生成高阶、长距、高平均适应度的模式，最终生成全局最优解。Holland 的模式定理通过计算有用相似性，即模式(Pattern)，奠定了遗传算法的数学基础。该定理是遗传算法的主要定理，在一定程度上解释了遗传算法的机理、数学特性以及很强的计算能力等特点。

## 3.1　模式定理

不失一般性，本节以二进制串作为编码方式来讨论模式定理(Pattern Theorem)。

**定义 3.1**　基于三值字符集{0，1，*}所产生的能描述具有某些结构相似性的 0、1 字符串集的字符串称作模式。

以长度为 5 的串为例，模式 *0001 描述了在位置 2、3、4、5 具有形式“0001”的所有字符串，即(00001，10001)。由此可以看出，模式的概念为我们提供了一种简洁的用于描述在某些位置上具有结构相似性的 0、1 字符串集合的方法。

引入模式后，我们看到一个串实际上隐含着多个模式(长度为 $n$ 的串隐含着 $2^n$ 个模式)，一个模式可以隐含在多个串中，不同的串之间通过模式而相互联系。遗传算法中串的运算实质上是模式的运算。因此，通过分析模式在遗传操作下的变化，就可以了解什么性质被延续，什么性质被丢弃，从而把握遗传算法的实质，这正是模式定理所揭示的内容。

**定义 3.2**　模式 $H$ 中确定位置的个数称作该模式的阶数，记作 $o(H)$。比如，模式 011*1* 的阶数为 4，而模式 0* * * * * 的阶数为 1。

显然，一个模式的阶数越高，其样本数就越少，因而确定性越高。

**定义 3.3**　模式 $H$ 中第一个确定位置和最后一个确定位置之间的距离称作该模式的定义距，记作 $\delta(H)$。比如，模式 011*1* 的定义距为 4，而模式 0* * * * * 的定义距为 0。

模式的阶数和定义距描述了模式的基本性质。

下面通过分析遗传算法的三种基本遗传操作对模式的作用来讨论模式定理。令 $A(t)$ 表示第 $t$ 代中串的群体，以 $A_j(t)(j=1，2，\cdots，n)$表示第 $t$ 代中第 $j$ 个个体串。

**1.** 选择算子

在选择算子作用下，与某一模式所匹配的样本数的增减依赖于模式的平均适应度，与

群体平均适应度之比，平均适应度高于群体平均适应度的将呈指数级增长；而平均适应度低于群体平均适应度的模式将呈指数级减少。其推导如下：

设在第 $t$ 代种群 $A(t)$ 中模式所能匹配的样本数为 $m$，记为 $m(H, t)$。在选择中，一个位串 $A_j$ 以概率 $P_j = f_j / \sum_i f_i$ 被选中并进行复制，其中 $f_j$ 是个体 $A_j(t)$ 的适应度。假设一代中群体大小为 $n$，且个体两两互不相同，则模式 $H$ 在第 $t+1$ 代中的样本数为

$$m(H, t+1) = m(H, t) n \frac{f(H)}{\sum_i f_i} \tag{3.1}$$

式中，$f(H)$ 是在 $t$ 时刻对应于模式的位串的平均适应度。令群体平均适应度为 $\bar{f} = \sum_i f_i / n$，则有

$$m(H, t+1) = m(H, t) \frac{f(H)}{\bar{f}} \tag{3.2}$$

现在，假定模式 $H$ 的平均适应度高于群体平均适应度，且设高出部分为 $c\bar{f}$，$c$ 为常数，则有

$$m(H, t+1) = m(H, t) \frac{\bar{f} + c\bar{f}}{\bar{f}} = (1 + c) m(H, t) \tag{3.3}$$

假设从 $t=0$ 开始，$c$ 保持为常值，则有

$$m(H, t+1) = m(H, 0)(1 + c) \tag{3.4}$$

**2．**交叉算子

然而仅有选择操作，并不能产生新的个体，即不能对搜索空间中新的区域进行搜索，因此引入了交叉操作。下面讨论模式在交叉算子作用下所发生的变化，这里我们只考虑单点交叉的情况。

模式 $H$ 只有当交叉点落在定义距之外才能生存。在简单交叉（单点交叉）下 $H$ 的生存概率 $P_s = 1 - \delta(H)/(t-1)$。例如，一个长度为 5 的串以及隐含其中的两个模式为

$A = 010110$

$H_1 = *1***0$

$H_2 = ***11*$

我们注意到模式 $H_1$ 的定义长度为 4，那么交叉点在 $6-1=5$ 个位置随机产生时，$H_1$ 遭破坏的概率 $P_d = \delta(H_2)/(m-1) = 1/5$，即生存概率为 4/5。

而交叉本身也是以一定的概率 $P_c$ 发生的，所以模式 $H$ 的生存概率为

$$P_s = 1 - P_c P_d = 1 - \frac{P_c \cdot \delta(H_2)}{m-1} \tag{3.5}$$

现在我们考虑交叉发生在定义距内，模式 $H$ 不被破坏的可能性。在前面的例子中，若与 $A$ 交叉的串在位置 2、6 上有一位与 $A$ 相同，则 $H_1$ 将被保留。考虑到这一点，式(3.5)给出的生存概率只是一个下界，即有

$$P_s \geq 1 - \frac{P_c \cdot \delta(H)}{m-1} \tag{3.6}$$

可见，模式在交叉算子作用下定义距短的模式将增多。

**3.** 变异算子

假定串的某一位置发生改变的概率为 $P_m$，则该位置不变的概率为 $1-P_m$，而模式 $H$ 在变异算子作用下若要不受破坏，则其中所有的确定位置（“0”或“1”的位）必须保持不变。因此模式 $H$ 保持不变的概率为 $(1-P_m)^{o(H)}$，其中 $o(H)$ 为模式 $H$ 的阶数。当 $P_m \ll 1$ 时，模式 $H$ 在变异算子作用下的生存概率为

$$P_s = (1-P_m)^{o(H)} \approx 1-o(H)P_m \tag{3.7}$$

综上所述，模式 $H$ 在遗传算子选择、交叉和变异的共同作用下，其子代的样本数为

$$m(H, t+1) \geqslant m(H, t)\frac{f(H)}{\bar{f}}\left[1-P_c\frac{\delta(H)}{l-1}\right][1-o(H)P_m] \tag{3.8}$$

式(3.8)忽略了极小项 $P_c \cdot \delta(H)/(l-1)+o(H)\cdot P_m$。通过式(3.8)，我们就可以给出模式定理。

定理 **3.1**(模式定理)　在遗传算子选择、交叉和变异的作用下，具有阶数低、长度短、平均适应度高于群体平均适应度的模式在子代中将以指数级增长。

统计学的研究表明：在随机搜索中，要获得最优的可行解，则必须保证较优解的样本呈指数级增长，而模式定理保证了较优的模式（遗传算法的较优解）的样本呈指数级增长，从而给出了遗传算法的理论基础。另外，由于遗传算法总能以一定的概率遍历到解空间的每一个部分，因此在选择算子的条件下总能得到问题的最优解。

## 3.2　积木块假设

由模式定理可知，具有阶数低、长度短、平均适应度高于群体平均适应度的模式在子代中将以指数级增长。这类模式在遗传算法中非常重要，在这一节给这些模式一个特别的名字——积木块(Building Block)。

定义 **3.4**　阶数低、长度短和适应度高的模式称为积木块。

假设 **3.1**(积木块假设(**Building Block Hypothesis**)　阶数低、长度短、适应度高的模式（积木块）在遗传算子作用下，相互结合，能生成阶数高、长度长、适应度高的模式，可最终生成全局最优解。

与积木块一样，一些好的模式在遗传算法操作下相互拼搭、结合，产生适应度更高的串，从而找到更优的可行解，这正是积木块假设所揭示的内容。下面用图来说明遗传算法中积木块生成最优解的过程。

假设每代种群规模为 8，$S_i$ 表示每代群体中第 $i$ 个个体，问题的最优解由积木块 AA、BB、CC 组成。图 3.1 所示为初始种群，个体 $S_1$、$S_7$ 含有 AA，个体 $S_4$、$S_8$ 含有 BB，个体 $S_3$ 含有 CC。

当种群进化一代后，第二代种群如图 3.2 所示，个体 $S_1$、$S_3$、$S_7$ 含有 AA，个体 $S_2$、$S_7$ 含有 BB，个体 $S_3$、$S_6$ 含有 CC。个体 $S_3$ 含有 AA、CC，个体 $S_7$ 含有 AA、BB。当种群进化到第二代后，第三代种群如图 3.3 所示，在群体中，出现了含有积木块 AA、BB、CC 的个体 $S_3$，个体 $S_3$ 就是问题的最优解。

| | | | |
|---|---|---|---|
| $S_1$ | AA | | |
| $S_2$ | | | |
| $S_3$ | | | CC |
| $S_4$ | | BB | |
| $S_5$ | | | |
| $S_6$ | | | |
| $S_7$ | AA | | |
| $S_8$ | | BB | |

图 3.1　初始种群

| | | | |
|---|---|---|---|
| $S_1$ | AA | | |
| $S_2$ | | BB | |
| $S_3$ | AA | | CC |
| $S_4$ | | | |
| $S_5$ | | | |
| $S_6$ | | | CC |
| $S_7$ | AA | BB | |
| $S_8$ | | | |

图 3.2　第二代种群

| | | | |
|---|---|---|---|
| $S_1$ | AA | | |
| $S_2$ | | BB | |
| $S_3$ | AA | BB | CC |
| $S_4$ | | | |
| $S_5$ | | | |
| $S_6$ | | BB | |
| $S_7$ | AA | | CC |
| $S_8$ | | | |

图 3.3　第三代种群

模式定理保证了较优的模式(遗传算法的较优解)样本数呈指数增长，从而满足了寻找最优解的必要条件，即遗传算法存在着寻找到全局最优解的可能性。而这里的积木块假设则指出，遗传算法具备找到全局最优解的能力，即积木块在遗传算子作用下，能生成阶数高、长度长、适应度高的模式，最终生成全局最优解。

## 3.3　欺骗问题

在遗传算法中，将所有妨碍适应度高的个体的生成从而影响遗传算法正常工作的问题统称为欺骗问题(Deceptive Problem)。遗传算法运行过程具有将阶数低、长度短、平均适应度高于群体平均适应度的模式重组成高阶模式的趋势。如果在低阶模式中包含了最优

解，则遗传算法就可能找出这个最优解来。但是低阶、高适应度的模式可能没有包含最优串的具体取值，于是遗传算法就会收敛到一个次优的结果。下面给出有关欺骗性的概念。

**定义 3.5**（竞争模式）　若模式 $H$ 和 $H'$ 中 * 的位置完全一致，但任一确定位的编码均不同，则称 $H$ 和 $H'$ 互为竞争模式。

**定义 3.6**（欺骗性）　假设 $f(X)$ 的最大值对应的 $X$ 的集合为 $X^*$，$H$ 为一包含 $X^*$ 的 $m$ 阶模式，$H$ 的竞争模式为 $H'$，而且 $f(H)>f(H')$，则 $f$ 为 $m$ 阶欺骗。

**定义 3.7**（最小欺骗性）　在欺骗问题中，为了造成骗局所需设置的最小的问题规模（即阶数）称为最小欺骗性 。其主要思想是在最大程度上违背积木块假设，是优于由平均的短积木块生成局部最优点的方法。这里的“最小”是指问题规模采用两位。下面是一个由 4 个阶数为 2、有 2 个确定位置的模式集：

* * *0* * * * *0*　$f(00)$

* * *0* * * * *1*　$f(01)$

* * *1* * * * *0*　$f(10)$

* * *1* * * * *1*　$f(11)$

为简单（达到最小）起见，我们不考虑 * 位，令 $f(11)$ 为全局最优解，为了欺骗遗传算法，Goldberg 设计了两种情况：

Type1：$f(01)>f(00)$

Type2：$f(00)>f(01)$

满足 $f(0*)>f(1*)$ 或者 $f(*0)>f(*1)$。

按 Holland 的模式定理，最小欺骗问题将给遗传算法造成很大困难，遗传算法甚至找不到最优解。但 Goldberg 实验的结果却是：Type1 问题基本上都很快找到了最优解，Type2 问题找到和找不到两种情况都可能出现。

遗传算法中欺骗性的产生往往与适应度函数确定和调整、基因编码方式选取相关。采用合适的编码方式或调整适应度函数，就可能化解和避免欺骗问题。下面以合适的编码方式为例来说明。

一个 2 位编码的适应度函数为

$$f(x)=4+\frac{11}{6}x-4x^2+\frac{7}{6}x^3$$

采用二进制编码，计算个体的函数值（见表 3.1），则存在第二类欺骗问题。采用格雷编码，计算个体的函数值（见表 3.2），则第二类欺骗问题化解为第一类欺骗问题。

表 **3.1**　二进制编码及函数值

| 编码 | 对应整数解 | 函数值 |
|---|---|---|
| 00 | 0 | 4 |
| 01 | 1 | 3 |
| 10 | 2 | 1 |
| 11 | 3 | 5 |

表 **3.2**　格雷编码及函数值

| 编码 | 对应整数解 | 函数值 |
|---|---|---|
| 00 | 0 | 4 |
| 01 | 1 | 3 |
| 11 | 2 | 1 |
| 10 | 3 | 5 |

采用适当的适应度函数调整方法，设

$$g(x):g(00)=128,\ g(01)=1,\ g(10)=g(11)=32$$

如果适应度函数 $f(x)=g(x)$，则

$$f(0*)=\frac{f(00)+f(01)}{2}=64.5$$

$$f(1*)=\frac{f(10)+f(11)}{2}=32$$

故存在欺骗问题。

如果用适应度函数的调整方法，$f(x)=\mathrm{lb}g(x)$，则

$$f(00)=7,\ f(01)=0,\ f(11)=f(10)=5$$

得 $f(0*)=3.5$，$f(1*)=5$，故不会产生欺骗问题。

## 3.4 遗传算法的未成熟收敛问题及其防止

在实际中，很多参数优化问题是多参数和非线性的，且往往还伴随着不可微和参数耦合等问题。这时用传统的优化方法解决问题，效率很低，有时候甚至根本得不出结果。遗传算法的高鲁棒性和有效性为解决这类问题提供了一种有效的途径。

在实际应用中，遗传算法显示出了顽强的生命力，以其突出的优点越来越受到重视。但是遗传算法也存在着缺点，在实际应用中也因此出现了一些问题，其中很重要的是遗传算法未成熟收敛(也称为早熟，Premature Convergence，简称 PC）问题。对于遗传算法的应用，解决未成熟收敛问题是必要的，否则，遗传算法的一些优良性能(例如全局寻优能力）将无法完全体现出来。

### 3.4.1 遗传算法的未成熟收敛问题

**1.** 未成熟收敛现象

未成熟收敛现象主要表现在两个方面：

(1) 群体中所有的个体都陷于同一极值而停止进化。

(2) 接近最优解的个体总是被淘汰，进化过程不收敛。

未成熟收敛现象是遗传算法中的特有现象，且十分常见。它指的是，当还未达到全局最优解或满意解时，群体中不能再产生性能超过父代的后代，群体中的各个个体非常相似。未成熟收敛的重要特征是群体中个体结构的多样性急剧减少，这将导致遗传算法的交叉算子和选择算子不能再产生更有生命力的新个体。遗传算法希望找到最优解或满意解，而不是在找到最优解或满意解之前，整个群体就收敛到一个非优个体，它希望能够保持群体中个体结构的多样性，从而使搜索能够进行下去。未成熟收敛问题就像其他算法中的局部极小值(极大值)问题，但它和局部极小值问题有着本质上的不同，未成熟收敛问题并不一定出现在局部极小点。同时，它的产生是带有随机性的，人们很难预见是否会出现未成熟收敛问题。

**2.** 未成熟收敛产生的主要原因

未成熟收敛产生的主要原因有以下几点：

(1) 理论上考虑的选择、交叉、变异操作都是绝对精确的，它们之间相互协调，能搜索

到整个解空间，但在具体实现时很难达到这个要求。

（2）所求解的问题是遗传算法欺骗问题。当解决的问题对于标准遗传算法来说比较困难时，遗传算法就会偏离寻优方向，这种问题被称为遗传算法欺骗问题。

（3）遗传算法处理的群体是有限的，因而存在随机误差，它主要包括取样误差和选择误差。取样误差是指所选择的有限群体不能代表整个群体所产生的误差。当表示有效模板串的数量不充分或所选的串不是相似子集的代表时，遗传算法就会发生上述类似的情况。小群体中的取样误差妨碍模板的正确传播，因而阻碍模板原理所预测的期望性能产生。选择误差是指不能按期望的概率进行个体选择。

对一个染色体来说，在遗传操作中只能产生整数个后代。在有限群体中，模板的样本不可能以任意精度反映所要求的比例，这是产生取样误差的根本原因，加上随机选择的误差，就可以导致模板样品数量与理论预测值有很大差别。随着这种偏差的积累，一些有用的模板将会从群体中消失。遗传学家认为当群体很小时，选择就不会起作用，这时有利的基因可能被淘汰，有害的基因可能被保留。引起群体结构发生变化的主要因素是随机波动的——遗传漂移，它也是产生未成熟收敛的一个主要原因。对此可以采用增大群体容量的方法来减缓遗传漂移，但这样做可能导致算法效率的降低。

上述三个方面都有可能产生未成熟收敛现象，即群体中个体的多样性过早地丢失，从而使算法陷入局部最优点。

在遗传算法处理过程的每个环节都有可能导入未成熟收敛的因素。遗传算法未成熟收敛产生的主要原因是，在迭代过程中，未得到最优解或满意解以前，群体就失去了多样性。具体表现在以下几个方面：

（1）在进化初始阶段，生成了具有很高适应度的个体 $X$。

（2）在基于适应度比例的选择下，其他个体被淘汰，大部分个体与 $X$ 一致。

（3）相同的两个个体进行交叉，从而未能生成新个体。

（4）通过变异所生成的个体适应度高但数量少，所以被淘汰的概率很大。

（5）群体中的大部分个体都处于与 $X$ 一致的状态。

### 3.4.2　未成熟收敛的防止

在分析了未成熟收敛产生的原因后，下面要解决的是如何防止该现象的发生，即如何维持群体多样性以保证在寻找到最优解或满意解以前，不会发生未成熟收敛现象。解决的方法有以下几种。

**1．**重新启动法

这是实际应用中最早出现的方法之一。在遗传算法搜索中碰到未成熟收敛问题而不能继续时，则随机选择一组初始值重新进行遗传算法操作。假设每次执行遗传算法后陷入不成熟收敛的概率为 $Q(0\leqslant Q<1)$，那么做 $n$ 次独立的遗传算法操作后，可避免未成熟收敛的概率为 $F(n)=1-Q^n$，随着 $n$ 的增大，$F(n)$将趋于 1。但是，对于 $Q$ 较大的情况，如果优化对象很复杂以及每次执行时间都很长，采用该办法显然是不合适的。

**2．**配对策略(**Mating Strategies** )

为了维持群体的多样性，可以有目的地选择配对个体。一般情况下，在物种的形成过程中要考虑配对策略，以防止根本不相似的个体进行配对。因为在生物界，不同种族之间

一般是不会杂交的，这是因为它们的基因结构不同，会发生互斥作用，同时杂交后会使种族失去其优良特性。因此，配对受到限制，即大多数是同种或近种相配，以使一个种族的优良特性得以保存和发扬。然而，这里所说的匹配策略有不同的目的。其目的是，由不同的父辈产生的个体试图比其父辈更具有多样性，Goldberg 的共享函数(Sharing Function)就是一种间接匹配策略。该策略对生物种(Species)内的相互匹配或至少对占统治地位的物种内的相互匹配有一定限制。Eshelman 提出了一种可以更直接地防止相似个体交配的方法——防止乱伦机制(Incest Prevention Mechanism)。参与交配的个体是随机配对的，但只有当参与配对的个体间的海明距离超过一定阈值时，才允许它们进行交配。最初的阈值可采用初始群体海明距离的期望平均值，随着迭代过程的发展，阈值可以逐步减小。尽管 Eshelman 的方法并不能明显地阻止同辈或相似父辈之间进行交配，但只要个体相似，它就有一定的影响。匹配策略是对具有一定差异的个体进行配对，这在某种程度上可以维持群体的多样性。但它同时也具有一定的副作用，即交叉操作会使较多的模板遭到破坏，只有较少的共享模板得以保留。

3．重组策略(**Recombination Strategies**)

重组策略就是使用交叉算子。在某种程度上，交叉操作试图产生与其父辈不同的个体，从而使产生的群体更具多样性。能使交叉操作更具有活力的最简单的方法就是，增加其使用的频率和使用动态改变适应度函数的方法，如共享函数方法。另一种方法是把交叉点选在个体的具有不同值的位上。只要父辈个体至少有两位不同，所产生的子代个体就会与其父辈不相同。维持群体多样性的更基本的方法是，使用更具有破坏性的交叉算子，如均匀交叉算子。该算子试图交叉近一半的不同位，因而保留的模板比单点或两点交叉所保留的模板要少得多。总之，重组策略主要是从使用频率和交叉点两方面考虑，来维持群体的多样性。这对采用随机选择配对个体进行交叉操作可能有特定的意义，但对成比例选择方式，效果则不一定明显。

4．替代策略(**Replacement Strategies** )

匹配策略和重组策略分别是在选择、交叉阶段，通过某种策略来维持群体的多样性。而替代策略是确定在选择、交叉产生的个体中，选择哪一个个体进入新一代群体。De Jong 采用排挤(Crowding)模式，用新产生的个体去替换父辈中类似的个体。Syscuda 和 Whiteley 也采用类似的方法，他们仅把与父辈各个个体均不相似的新个体添加到群体中。这种替换策略仅从维持群体的多样性出发，存在一定的负面影响，即交叉操作会破坏较多模板，但这种影响比前两种策略的要少。

## 3.5 性能评估

遗传算法的实现涉及到它的五个要素：参数编码、初始群体的设定、适应度函数的设计、遗传操作设计和控制参数设定，而每个要素又对应不同的环境，存在各种相应的设计策略和方法。不同的策略和方法决定了各自的遗传算法具有不同的性能或特征。因此，评估遗传算法的性能对于研究和应用遗传算法是十分重要的。

目前，遗传算法的评估指标大多采用适应度值。特别在没有具体要求的情况下，一般采用各代中最优个体的适应度值和群体的平均适应度值。以此为依据，De Jong 提出了两

个用于定量分析遗传算法的测度：离线性能（Off-line Performance）测度和在线性能（On-line Performance）测度，得到了两个评估准则。

**1.** 在线性能评估准则

定义 **3.8**　设 $X_e(s)$为环境 $e$ 下策略 $s$ 的在线性能，$f_e(t)$为时刻 $t$ 或第 $t$ 代中相应于环境 $e$ 的目标函数或平均适应度函数，则 $X_e(s)$可以表示为

$$X_e(s) = \frac{1}{T}\sum_{t=1}^{T} f_e(t) \tag{3.9}$$

式(3.9) 表明，在线性能可以用从第一代到当前的优化进程的平均值来表示。

**2.** 离线性能评估准则

定义 **3.9**　设 $X_e^*(s)$为环境 $e$ 下策略 $s$ 的离线性能，则有

$$X_e^*(s) = \frac{1}{T}\sum_{t=1}^{T} f_e^*(t) \tag{3.10}$$

式中，$f_e^*(t)=best\{f_e(1), f_e(2), \cdots, f_e(t)\}$。

式(3.10) 表明，离线性能是特定时刻或特定代的最佳性能的累积平均。具体地说，在进化过程中，每进化一代就统计目前为止的各代中的最佳适应度或最佳平均适应度，并计算对进化代数的平均值。

De Jong 指出：离线性能用于测量算法的收敛性，在应用时，优化问题的求解可以得到模拟，在一定的优化进程停止准则下，当前最好的解可以被保存和利用；而在线性能用于测量算法的动态性能，在应用时，优化问题的求解必须通过真实的实验在线实现，可以迅速得到较好的优化结果。但是，从遗传算法的运行机理可知，在遗传算子的作用下，群体的平均适应度呈现增长的趋势，因此，定义 3.8 和定义 3.9 中的 $f_e(t)$和 $f_e^*(t)$相差不大，它们所反映的性质也基本一样。

下面以最优化方法的收敛速度和收敛准则来讨论遗传算法的性能。

一般优化问题可描述为求 $x=(x_1, x_2, \cdots, x_n)^{\mathrm{T}}$，使 $f(x)$达到最小（或最大）。最优化方法通常采用迭代方法求它的最优解，其基本思想为：给定一个初始点 $x_0$，按照某一迭代规则产生一个点列$\{x_i\}$，使得当$\{x_i\}$为有穷点列时，其最后一个点是最优化问题的最优解。当$\{x_i\}$是无穷点列时，它有极限点，且其极限点是最优化问题的最优解。一个好的优化算法为：当 $x_i$ 能稳定地接近全局极小点（或极大点）的邻域时，迅速收敛于 $x^*$，当满足给定的收敛准则时，迭代终止。

假设一算法产生的迭代点列$\{x_i\}$在某种范数意义下收敛，即

$$\lim_{i\to\infty} \| x_i - x^* \| = 0 \tag{3.11}$$

式中，$x_{i+1}=x_i+\alpha_i$，$\alpha_i$ 为步长因子。若存在实数 $\alpha>0$ 及一个与迭代次数无关的常数 $q>0$，使得

$$\lim_{i\to\infty} \frac{\| x_{i+1} - x^* \|}{\| x_i - x^* \|} = q \tag{3.12}$$

则称此算法产生的迭代点列$\{x_i\}$具有 $q-\alpha$ 阶收敛速度（$\alpha$ 为迭代步长因子）。因为$\| x_{i+1}-x_i \|$是$\| x_i-x^* \|$的一个估计，所以在实际中，一般用$\| x_{i+1}-x_i \|$代替$\| x_i-x^* \|$，作为迭代终止判决条件。

① 当 $\alpha=1$，$q>0$ 时，称$\{x_i\}$具有 $q$ 线性收敛速度。

② 当 $1<\alpha<2$，$q>0$ 或 $\alpha=1$，$q=0$ 时，称$\{x_i\}$具有 $q$ 超线性收敛速度。

③ 当 $\alpha=2$ 时，称$\{x_i\}$具有 $q$ 二阶收敛速度。

具有超线性收敛速度和二阶收敛速度的迭代算法收敛比较快。

关于算法的终止准则,实际应用中可以用各种不同的方法来确定收敛准则。Himmeblau 提出了下面的终止准则。

当 $\| x_i \| > \varepsilon$ 和 $| f(x_i) | > \varepsilon$ 时，采用

$$\frac{\| x_{i+1} - x_i \|}{\| x_i \|} \leqslant \varepsilon \text{ 或 } \frac{| f(x_{i+1}) - f(x_i) |}{| f(x_i) |} \leqslant \varepsilon \tag{3.13}$$

否则采用

$$| x_{i+1} - x_i | < \varepsilon \text{ 或 } | f(x_{i+1}) - f(x_i) | < \varepsilon \tag{3.14}$$

式中，$\varepsilon$ 为根据实际问题要求精度给出的适当小的正数。

根据以上 Himmeblau 提出的终止准则，实际中可以用各代适应度函数的均值之差来衡量遗传算法的收敛特性。定义收敛性测量函数为

$$C_e(s) = \frac{1}{T}\sum_{t=1}^{T}[f_e(t+1) - f_e(t)] \tag{3.15}$$

式中，$f_e(t)$为时刻 $t$ 或第 $t$ 代中相应于环境 $e$ 的目标函数或平均适应度函数。

从优化问题中寻找最优解或最优解组的角度考虑，可以定义部分在线特性：

$$f_n(s) = \frac{1}{T}\sum_{t=1}^{T} f'_e(t) \tag{3.16}$$

式中，$f'_e(t)$为群体中对应于最优解或最优解组的个体适应度的均值。

## 3.6 小生境技术和共享函数

Cavicchio 提出只有在子串的适应度超过父代的情况下，子串才能替代父串进入下一代群体的预选择机制，该机制趋向于替换与其本身相似的个体，能够维持群体的分布特性，并且不断地以优秀个体来更新种群，使种群不断被优化。

De Jong 提出基于排挤的机制，其思想来源于一个有趣的生物现象：在一个有限的生存空间中，各种不同的生物为了延续生存，必须相互竞争各种有限的生存资源。差别较大的个体由于生活习性不同而很少竞争。处于平衡状态的大小固定的种群，新生个体将代替与之相似的旧个体。排挤机制用海明距离来度量个体之间的相似性。排挤机制可以维护当前种群的多样性。这就是小生境技术。

Goldberg 和 Richardson 利用共享函数来度量两个个体的相邻关系和程度。给定个体 $g$，它的共享函数由它与种群中其他个体的相似程度决定。将 $g$ 与种群中其他个体逐个比较，若很相似，则对 $g$ 的共享函数加一个较大值；否则，就加一个较小值。个体共享度为该个体与群体内其他个体共享函数值之和，即

$$S_i = \sum_{j=1}^{n} S(d_{ij}) \tag{3.17}$$

式中，$d_{ij}$为个体$i$和个体$j$之间的关系亲密程度；$S$为共享函数；$S_i$为个体$i$在群体中的共享度。

个体的适应度的调整公式为

$$f_s(i) = \frac{f(i)}{S_i} \tag{3.18}$$

# 第四章　遗传算法的基本原理与方法

遗传算法是一种基于生物进化原理构想出来的搜索最优解的仿生算法，它模拟基因重组与进化的自然过程，把待解决问题的参数编成二进制码或十进制码(也可编成其他进制码)即基因，若干基因组成一个染色体(个体)，许多染色体进行类似于自然选择、配对交叉和变异的运算，经过多次重复迭代(即世代遗传)直至得到最后的优化结果。习惯上，适应度值越大，表示解的质量越好。对于求解最小值问题，可通过变换转为求解最大值问题。遗传算法以群体为基础，不是以单点搜索为基础，能同时从不同点获得多个极值，因此不易陷入局部最优；遗传算法是对问题变量的编码集进行操作，而不是变量本身，有效地避免了对变量的微分操作运算；遗传算法只是利用目标函数来区别群体中的个体的好坏而不必对其进行过多的附加操作。本章将讨论遗传算法的实现涉及的六个主要因素：参数的编码，初始群体的设定，适应度函数的设计，遗传操作，算法控制参数的设定和约束条件的处理。

## 4.1　编　　码

编码是应用遗传算法时要解决的首要问题，也是设计遗传算法时的一个关键步骤。在遗传算法执行过程中，对不同的具体问题进行编码，编码的好坏直接影响选择、交叉、变异等遗传运算。那么，什么是编码呢？

在遗传算法中如何描述问题的可行解，即把一个问题的可行解从其解空间转换到遗传算法所能处理的搜索空间的转换方法就称为编码。而由遗传算法解空间向问题空间的转换称为解码(或称译码，Decoding)。

遗传算法的编码就是解的遗传表示，它是应用遗传算法求解问题的第一步。传统的二进制编码是 0、1 字符构成的固定长度串。二进制编码的一个缺点是汉明悬崖(Hamming Cliff)，就是在某些相邻整数的二进制代码之间有很大的汉明距离，使得遗传算法的交叉和突变都难以跨越。为克服此问题而提出的格雷码(Gray Code)，在相邻整数之间汉明距离都为 1。然而汉明距离在整数之间的差并非单调增加，引入了另一层次的隐悬崖。

De Jong 依据模式定理，提出了较为客观明确的编码准则：有意义的积木块编码规则，即所定编码应当易于生成与所求问题相关的短距和低阶的积木块；最小字符集编码规则，所定编码应采用最小字符集以使问题得到自然的表示或描述。近年来一些学者从理论上证

明了推导最小字符集编码规则时存在的错误，指出了大符号集编码可提供更多的模式，研究了二进制编码和十进制编码在搜索能力和保持群体稳定性上的差异，结果表明二进制编码比十进制编码搜索能力强，但不能保持群体稳定性。

针对二进制编码的遗传算法进行函数优化时精度不高的特点，Schraldolph 等提出了动态参数编码(Dynamic Parameter Coding)。为了得到较高精度，让遗传算法从很粗糙的精度开始收敛，当遗传算法找到一个区域后，就将搜索限制在这个区域，重新编码，重新启动，重复这一过程，直到达到要求的精度为止。

### 4.1.1 编码方法

针对一个具体应用问题，如何设计一种完美的编码方案一直是遗传算法的应用难点之一，也是遗传算法的一个重要研究方向。由于遗传算法应用的广泛性，迄今为止人们已经提出了许多种不同的编码方法，总的来说，可以分为三大类：二进制编码方法、符号编码方法和浮点数编码方法。下面介绍几种主要的编码方法。

**1. 二进制编码方法**

二进制编码方法是遗传算法中最主要的一种编码方法，它使用的编码符号集是由二进制符号 0 和 1 所组成的二值符号集{0，1}，它所构成的个体基因型是一个二进制编码符号串。二进制编码符号串的长度与问题所要求的求解精度有关。

二进制编码有以下优点：

(1) 编码、解码操作简单易行。

(2) 交叉、变异等遗传操作便于实现。

(3) 符合最小字符集编码原则。

(4) 便于利用模式定理对算法进行理论分析，因为模式定理是以二进制编码为基础的。

二进制编码有以下缺点：

首先，二进制编码存在着连续函数离散化时的映射误差。个体编码串的长度较短时，可能达不到精度的要求，而个体编码串的长度较大时，虽然能提高编码精度，但却会使遗传算法的搜索空间急剧扩大。其次，它不能直接反映出所求问题的本身结构特征，这样也就不便于开发针对问题的专门知识的遗传运算算子，很难满足积木块编码原则。

**2. 格雷码编码**

二进制编码不便于反映所求问题的结构特征，对于一些连续函数的优化问题等，也由于遗传运算的随机特性而使得其局部搜索能力较差。为了改进这个特性，人们提出用格雷码(Gray Code)来对个体进行编码。格雷码是这样一种编码方法，其连续的两个整数所对应的编码之间仅仅只有一个码位是不同的，其余码位都完全相同。格雷码是二进制编码方法的一种变形。表 4.1 所示为十进制数 0～15 的二进制码和相应的格雷码。

格雷码的主要优点是：

(1) 便于提高遗传算法的局部搜索能力。

(2) 交叉、变异等遗传操作便于实现。

(3) 符合最小字符集编码原则。

(4) 便于利用模式定理对算法进行理论分析。

表 4.1　二进制码和格雷码

| 十进制数 | 二进制码 | 格雷码 |
| --- | --- | --- |
| 0 | 0000 | 0000 |
| 1 | 0001 | 0001 |
| 2 | 0010 | 0011 |
| 3 | 0011 | 0010 |
| 4 | 0100 | 0110 |
| 5 | 0101 | 0111 |
| 6 | 0110 | 0101 |
| 7 | 0111 | 0100 |
| 8 | 1000 | 1100 |
| 9 | 1001 | 1101 |
| 10 | 1010 | 1111 |
| 11 | 1011 | 1110 |
| 12 | 1100 | 1010 |
| 13 | 1101 | 1011 |
| 14 | 1110 | 1001 |
| 15 | 1111 | 1000 |

**3.** 浮点数编码方法

对于一些多维、高精度要求的连续函数优化问题，使用二进制编码来表示个体时将会有一些不利之处。人们在一些经典优化算法的研究中所总结出的一些宝贵经验也就无法在这里加以利用，也不便于处理非平凡约束条件。为了克服二进制编码方法的缺点，人们提出了个体的浮点数编码方法。所谓浮点数编码方法，是指个体的每个基因值用某一范围内的一个浮点数来表示，个体的编码长度等于其决策变量的个数。因为这种编码方法使用的是决策变量的真实值，所以浮点数编码方法也叫做真值编码方法。

浮点数编码方法有以下几个优点：

(1) 适合于在遗传算法中表示范围较大的数。

(2) 适合于精度要求较高的遗传算法。

(3) 便于较大空间的遗传搜索。

(4) 改善了遗传算法的计算复杂性，提高了运算效率。

(5) 便于遗传算法与经典优化方法的混合使用。

(6) 便于设计针对问题的专门知识的知识型遗传算子。

(7) 便于处理复杂的决策变量约束条件。

**4.** 多参数级联编码

一般常见的优化问题中往往含有多个决策变量。对这种含有多个变量的个体进行编码的方法就称为多参数编码方法。多参数编码的一种最常用和最基本的方法是：将各个参数

分别以某种编码方法进行编码，然后再将它们的编码按照一定顺序连接在一起就组成了表示全部参数的个体编码。这种编码方法称为多参数级联编码方法。

在进行多参数级联编码时，每个参数的编码方式可以是二进制编码方法、格雷码、浮点数编码或符号编码等编码方式中的一种，每个参数可以具有不同的上下界，也可以有不同的编码长度或编码精度。

**5.** 多参数交叉编码

多参数交叉编码方法的基本思想是：将各个参数中起主要作用的码位集中在一起，这样它们就不易于被遗传算子破坏掉。在进行多参数交叉编码时，可先对各个参数进行分组编码；然后取各个参数编码串中的最高位连接在一起，以它们作为个体编码串的前 $N$ 位编码，再取各个参数编码串中的次高位连接在一起，以它们作为个体编码串的第二组 $N$ 位编码……取各个参数编码串中的最后一位连接在一起，以它们作为个体编码串的最后几位。这样组成的长度为 $M\times N$ 位的编码串就是一个交叉编码串。

其他常用的编码技术有一维染色体编码、二维染色体编码、可变染色体长度编码和树结构编码。

### 4.1.2 编码评估策略

目前尚无一套既严格又完整的指导理论及评价标准来帮助我们设计编码方案，必须具体问题具体分析，采用不同的编码方法。评估编码策略常采用以下三个规范：

(1) 完备性(Completeness)：问题空间中的所有点(候选解)都能作为遗传算法空间中的点(染色体)表现。

(2) 健全性(Soundness)：遗传算法空间中的染色体能对应所有问题空间中的候选解。

(3) 非冗余性(Non-redundancy)：染色体和候选解一一对应。

应该注意，严格满足上述规范的编码方法和提高遗传算法的效率并无关系。在有些场合，允许生成致死基因的编码，虽然会导致冗余的搜索，但总的计算量可能反而减少，从而可以更有效地找出最优解。上述的三个策略虽然具有普遍意义，但是缺乏具体的指导思想，特别是满足这些规范的编码设计不一定能有效地提高遗传算法的搜索效率。相比之下，De Jong 提出了较为客观明确的编码评估准则，包括两个规则。

(1) 有意义积木块编码规则：所定编码应易于生成与所求问题相关的短距和低阶的积木块。

(2) 最小字符集编码规则：所定编码应采用最小字符集以使问题得到自然的表示或描述。

规则(1)基于模式定理和积木块假设，规则(2)提供了一种更为实用的编码原则。

## 4.2 选　　择

选择(Selection)又称复制(Reproduction)，是在群体中选择生命力强的个体产生新的群体的过程。遗传算法使用选择算子(又称为复制算子，Reproduction Operator)来对群体中的个体进行优胜劣汰操作：根据每个个体的适应度值大小选择，适应度较高的个体被遗传到下一代群体中的概率较大；适应度较低的个体被遗传到下一代群体中的概率较小。这

样就可以使得群体中个体的适应度值不断接近最优解。选择操作建立在对个体的适应度进行评价的基础之上。选择操作的主要目的是为了避免有用遗传信息的丢失，提高全局收敛性和计算效率。选择算子确定的好坏，直接影响到遗传算法的计算结果。选择算子确定不当，会造成群体中相似度值相近的个体增加，使得子代个体与父代个体相近，导致进化停止不前；或使适应度值偏大的个体误导群体的发展方向，使遗传失去多样性，产生早熟问题。

遗传算法中的选择操作就是用来确定如何从父代群体中按某种方法选取哪些个体遗传到下一代群体中的一种遗传运算，用来确定重组或交叉个体，以及被选个体将产生多少个子代个体。选择操作的策略与编码方式无关。表 4.2 所示为选择操作算子。

表 **4.2** 选择操作算子

| 序号 | 名　称 | 特　点 | 备　注 |
|---|---|---|---|
| 1 | 轮盘赌选择 | 选择误差较大 | GA 成员 |
| 2 | 随机竞争选择 | 比轮盘赌选择较好 | |
| 3 | 最佳保留选择 | 保证迭代终止结果为历代最高适应度个体 | |
| 4 | 无回放随机选择 | 降低选择误差，复制数小于 $f/(f+1)$，操作不方便 | |
| 5 | 确定性选择 | 选择误差更小，操作简易 | |
| 6 | 柔性分段复制 | 有效防止基因缺失，但需要选择参数 | |
| 7 | 自适应柔性分段式动态群体选择 | 群体自适应变化，提高搜索效率 | |
| 8 | 无回放式余数随机选择 | 误差最小 | 应用较广 |
| 9 | 均匀排序 | 与适应度大小差异程度正负无关 | |
| 10 | 稳态复制 | 保留父代中一些高适应度的串 | |
| 11 | 随机联赛选择 | | |
| 12 | 复制评价 | | |
| 13 | 最优保存策略 | 全局收敛，提高搜索效率，但不宜于非线性强的问题 | |
| 14 | 排挤选择 | 提高群体的多样性 | |
| 15 | 最优保存策略 | 保证全局收敛 | |

下面介绍几种常用的选择算子。

**1.** 轮盘赌选择

轮盘赌选择方法(Roulette Wheel Selection)是一种回放式随机采样方法。所有选择是从当前种群中根据个体的适应度值，按某种准则挑选出好的个体进入下一代种群。选择算子有多种，经典遗传算法中常采用的是轮盘赌(Wheel，或是比例选择 Proportional Selection)的选择方法，每个个体进入下一代的概率就等于它的适应度值与整个种群中个体适应度值和的比例，适应度值越高，被选中的可能性就越大，进入下一代的概率就越大。每个个体就像圆盘中的一个扇形部分，扇面的角度和个体的适应度值成正比，随机拨动圆盘，当圆盘停止转动时指针所在扇面对应的个体被选中，轮盘赌式的选择方法由此得名。

由于群体规模有限和随机操作等原因，使得个体实际被选中的次数与它应该被选中的期望值 $n \cdot f(x_i) / \sum_{i=1}^{n} f(x_i)$ 之间可能存在着一定的误差，因此这种选择方法的选择误差比较大，有时甚至连适应度较高的个体也选不上。图 4.1 所示为轮盘赌选择示意图。

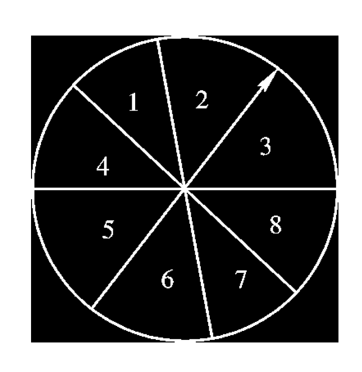



图 4.1　轮盘赌选择示意图

**2.** 随机竞争选择

随机竞争(Stochastic Tournament)选择与轮盘赌选择基本一样。在随机竞争选择中，每次按轮盘赌选择机制选取一对个体，然后让这两个个体进行竞争，适应度高的被选中，如此反复，直到选满为止。

**3.** 最佳保留选择

首先按轮盘赌选择方法执行遗传算法的选择操作，然后将当前群体中适应度最高的个体结构完整地复制到下一代群体中。其主要优点是能保证遗传算法终止时得到的最后结果是历代出现过的最高适应度的个体。

**4.** 无回放随机选择

这种选择操作方法也叫做期望值选择(Expected Value Selection)方法，它的基本思想是：根据每个个体在下一代群体中的生存期望值来进行随机选择运算。其具体操作过程是：

(1) 计算群体中每个个体在下一代群体中的生存期望数目 $N$。

(2) 若某一个体被选中参与交叉运算，则它在下一代中的生存期望数目减去 0.5；若某一个体未被选中参与交叉运算，则它在下一代中的生存期望数目减去 1.0。

(3) 随着选择过程的进行，若某一个体的生存期望数目小于 0 时，则该个体就不再有机会被选中。

这种选择操作方法能够降低一些选择误差，但操作不太方便。

**5.** 确定式选择

确定式选择方法的基本思想是：按照一种确定的方式来进行选择操作。具体操作过程是：

(1) 计算群体中各个个体在下一代群体中的期望生存数目 $N$。

(2) 用 $N$ 的整数部分确定各个对应个体在下一代群体中的生存数目。

(3) 用 $N$ 的小数部分对个体进行降序排序，顺序取前 $M$ 个个体加入到下一代群体中。至此可完全确定出下一代群体中 $M$ 个个体。

**6.** 无回放余数随机选择

无回放余数随机选择算法的选择操作可确保适应度比平均适应度大的一些个体能够被

遗传到下一代群体中，所以它的选择误差比较小。

**7.** 均匀排序

前面介绍的选择操作方法中，其选择依据主要是各个个体适应度的具体数值，一般要求它取非负值，这就使得我们在选择操作之前必须先对负的适应度进行变换处理。而排序(Ranking)选择算法的主要着眼点是个体适应度之间的大小关系，对个体适应度取正值或负值以及个体适应度之间的数值差异程度并无特别要求。

排序选择方法的思想是：对群体中的所有个体按其适应度大小进行排序，基于这个排序来分配各个个体被选中的概率。其具体操作过程是：

（1）对群体中的所有个体按其适应度大小进行降序排序。

（2）根据具体求解问题，设计一个概率分配表，将各个概率值按上述排列次序分配给各个个体。

（3）以各个个体所分配到的概率值作为其能够被遗传到下一代的概率，基于这些概率值用比例选择的方法来产生下一代群体。例如，表 4.3 所示为进行排序选择时所设计的一个概率分配表。由该表可以看出，各个个体被选中的概率只与其排列序号所对应的概率值有关，即只与个体适应度之间的大小次序有关，而与其适应度的具体数值无直接关系。

表 **4.3** 概率分配表

| 个体排列序号 | 适应度 | 选择概率 |
|---|---|---|
| 1 | 108 | 0.25 |
| 2 | 90 | 0.19 |
| 3 | 88 | 0.17 |
| 4 | 55 | 0.15 |
| 5 | 51 | 0.10 |
| 6 | 10 | 0.08 |
| 7 | －10 | 0.03 |
| 8 | －50 | 0.03 |

该方法的实施必须根据对所研究问题的分析和理解情况预先分配表，这个设计过程无一定规律可循。另一方面，虽然依据个体适应度之间的大小次序给各个个体分配了一个选中概率，但由于具体选中哪一个个体仍是使用了随机性较强的比例选择方法，因此排序选择方法仍具有较大的选择误差。

**8.** 最优保存策略

在遗传算法中，通过对个体进行交叉、变异等遗传操作而不断产生出新的个体。虽然随着群体的进化过程会产生出越来越多的优良个体，但由于选择、交叉、变异等操作的随机性，它们也有可能破坏掉当前群体中适应度最好的个体。而这却不是我们所希望发生的，因为它会降低群体的平均适应度，并且对遗传算法的运行效率、收敛性都有不利的影响，所以，我们希望适应度最好的个体要尽量保留到下一代群体中。为达到这个目的，可以使用最优保存策略进化模型来进行优胜劣汰操作，即当前群体中适应度最高的个体不参与交叉运算和变异运算，而是用它来替换掉本代群体中经过交叉、变异等操作后所产生的

适应度最低的个体。

最优保存策略进化模型的具体操作过程是：

(1) 找出当前群体中适应度最高的个体和适应度最低的个体。

(2) 若当前群体中最佳个体的适应度比总的迄今为止的最好个体的适应度还要高，则以当前群体中的最佳个体作为新的迄今为止的最好个体。

(3) 用迄今为止的最好个体替换掉当前群体中的最差个体。最优保存策略可视为选择操作的一部分。该策略的实施可保证迄今为止所得到的最优个体不会被交叉、变异等遗传运算所破坏，它是遗传算法收敛性的一个重要保证条件。但另一方面，它也容易使得某个局部最优个体不易被淘汰掉反而快速扩散，从而使得算法的全局搜索能力不强。所以，该方法一般要与其他一些选择操作方法配合起来使用，方可有良好的结果。

**9.** 随机联赛选择

随机联赛选择也是一种基于个体适应度之间大小关系的选择方法。其基本思想是每次选取几个个体中适应度最高的一个个体遗传到下一代群体中。在联赛选择操作中，只有个体适应度之间的大小比较运算，而无个体适应度之间的算术运算，所以它对个体适应度是取正值还是取负值无特别要求。联赛选择中，每次进行适应度大小比较的个体数目称为联赛规模。一般情况下，联赛规模 $N$ 的取值为 2。

联赛选择的具体操作过程是：

(1) 从群体中随机选取 $N$ 个个体进行适应度大小的比较，将其中适应度高的个体遗传到下一代群体中。

(2) 将上述过程重复 $M$ 次，就可以得到下一代群体中的 $M$ 个个体。

**10.** 排挤选择

De Jong 提出了排挤选择(Crowing Selection)方法，采用该方法，使得新生成的子代将替代或排挤相似的旧父代个体，提高群体的多样性。在采用覆盖群体模式的情况下(代沟为 0.1)，该方法可描述为：

(1) 设定排挤参数 CF。

(2) 从群体中随机地挑选 CF 个个体组成个体集(新的个体不包括在内)。

(3) 从这个集中淘汰一个个体，该个体与新个体的海明距离最短。

以上介绍的选择方法是常用的方法。每种方法对于遗传算法的在线和离线性能的影响各不相同。在具体使用时，应根据问题求解特点采用较合适的方法或者将几种方法结合使用。

## 4.3 交　　叉

在生物的自然进化过程中，两个同源染色体通过交配而重组，形成新的染色体，从而产生出新的个体或物种。交配重组是生物遗传和进化过程中的一个主要环节。模仿这个环节，遗传算法中使用交叉算子来产生新的个体。交叉(Crossover)又称重组(Recombination)，是按较大的概率从群体中选择两个个体，交换两个个体的某个或某些位。交叉运算产生子代，子代继承了父代的基本特征。交叉算子的设计包括如何确定交叉点位置和如何进行部分基因交换两个方面的内容。遗传算法中所谓的交叉运算，是指对两个相互配对的染色体按某

种方式相互交换其部分基因，从而形成两个新的个体。交叉运算是遗传算法区别于其他进化运算的重要特征，它在遗传算法中起着关键作用，是产生新个体的主要方法。

遗传算法中，在交叉运算之前还必须先对群体中的个体进行配对。目前常用的配对算法策略是随机配对，即将群体中的 $M$ 个个体以随机的方式组成$[M/2]$对配对个体组，其中$[X]$表示不大于 $X$ 的最大整数。交叉操作是在这些配对个体组中的两个个体之间进行的。表 4.4 所示为交叉操作。

表 **4.4** 交 叉 操 作

| 序号 | 名　　称 | 特　　点 | 适用编码 |
|---|---|---|---|
| 1 | 单点交叉 | 标准遗传算法成员 | 符号 |
| 2 | 两点交叉 | 使用较多 | 符号 |
| 3 | 均匀交叉 | 每一位以相同概率交叉 | 符号 |
| 4 | 多点交叉 | 交叉点大于 2 | 符号 |
| 5 | 部分匹配交叉 | | 序号 |
| 6 | 顺序交叉 | | 序号 |
| 7 | 循环交叉 | | 序号 |
| 8 | 启发式交叉 | 应用领域知识 | 序号 |
| 9 | 基于位置交换 | | 序号 |
| 10 | 算术交换 | | 序号 |

交叉算子的设计和实现与具体问题密切相关，并要和编码设计一同考虑，主要应包括交叉点的位置和如何交换部分基因。下面介绍几种适合于二进制编码个体或浮点数编码个体的交叉算子。

**1.** 单点交叉

单点交叉(One-point Crossover)又称为简单交叉，它是指在个体编码串中只随机设置一个交叉点，然后在该点相互交换两个配对个体的部分染色体。单点交叉的具体执行过程如下：

(1) 对个体进行两两随机配对，若群体大小为 $M$，则共有$[M/2]$对相互配对的个体组。

(2) 对每一对相互配对的个体，随机设置某一基因座之后的位置为交叉点，若染色体的长度为 $N$，则共有 $N-1$ 个可能的交叉点位置。

(3) 对每一对相互配对的个体，依设定的交叉概率在其交叉点处相互交换两个个体的部分染色体，从而产生出两个新的个体。

图 4.2 所示为单点交叉运算的示意图。




图 4.2　单点交叉运算的示意图

**2．两点交叉与多点交叉**

两点交叉(Two-point Crossover) 是指在个体编码串中随机设置了两个交叉点，然后再进行部分基因交换。两点交叉的具体操作过程是：

(1) 在相互配对的两个个体编码串中随机设置两个交叉点。

(2) 交换两个个体在所设定的两个交叉点之间的部分染色体。

图 4.3 所示为两点交叉运算的示意图。




图 4.3　两点交叉运算的示意图

将单点交叉与两点交叉的概念加以推广，可得到多点交叉(Multi-point Crossover) 的概念。多点交叉是指在个体编码串中随机设置多个交叉点，然后进行基因交换。多点交叉又称为广义交叉，其操作过程与单点交叉和两点交叉相类似。图 4.4 所示为有三个交叉点时的交叉操作示意图。

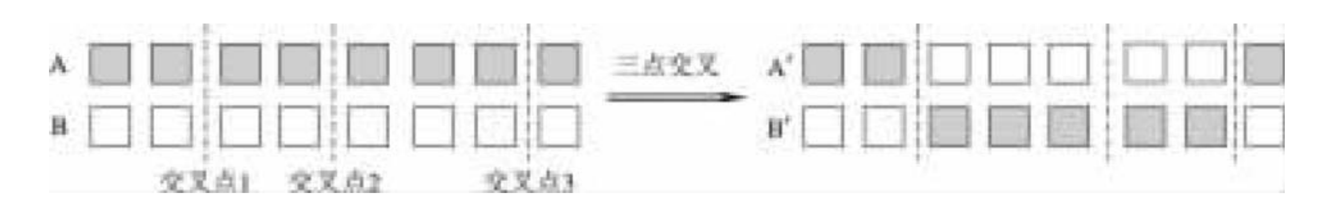



图 4.4　三点交叉运算的示意图

需要说明的是，一般情况下不使用多点交叉算子，因为它有可能破坏一些好的模式。事实上，随着交叉点数的增多，个体的结构被破坏的可能性也逐渐增大，这样就很难有效地保存较好的模式，从而影响遗传算法的性能。

**3．均匀交叉**

均匀交叉(也称为一致交叉，Uniform Crossover)是指两个配对个体的每个基因座上的基因都以相同的交叉概率进行交换，从而形成两个新的个体。均匀交叉实际上可归属于多点交叉的范围，其具体运算可通过设置一屏蔽字来确定新个体的各个基因如何由哪一个父代个体来提供。均匀交叉的主要操作过程如下：

(1) 随机产生一个与个体编码串长度等长的屏蔽字 $W=\omega_1\omega_2\cdots\omega_i\cdots\omega_L$，其中 $L$ 为个体编码串长度。

(2) 由下述规则从 A、B 两个父代个体中产生出两个新的子代个体 A′、B′。

① 若 $\omega_i=0$，则 A′在第 $i$ 个基因座上的基因值继承 A 的对应基因值，B′在第 $i$ 个基因座上的基因值继承 B 的对应基因值。

② 若 $\omega_i=1$，则 A′在第 $i$ 个基因座上的基因值继承 B 的对应基因值，B′在第 $i$ 个基因座上的基因值继承 A 的对应基因值。

均匀交叉操作的示例如图 4.5 所示。

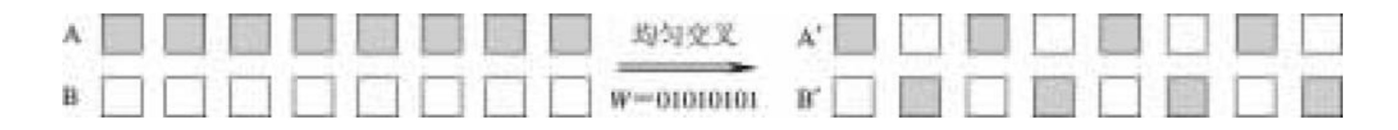



图 4.5　均匀交叉运算的示意图

**4. 算术交叉**

算术交叉(Arithmetic Crossover)是指由两个个体的线性组合而产生出两个新的个体。为了能够进行线性组合运算，算术交叉的操作对象一般是由浮点数编码所表示的个体。

假设在两个个体 $X_A'$、$X_B'$ 之间进行算术交叉，则交叉运算后所产生的两个新个体为

$$\begin{cases} X_A^{t+1} = \alpha X_B^t + (1-\alpha) X_A^t \\ X_B^{t+1} = \alpha X_A^t + (1-\alpha) X_B^t \end{cases} \tag{4.1}$$

其中，α 为一个参数，α 可以是一个常数(此时所进行的交叉运算称为均匀算术交叉)，α 也可以是一个由进化代数所决定的变量(此时所进行的交叉运算称为非均匀算术交叉)。

算术交叉的主要操作过程为：

(1) 确定两个个体进行线性组合时的系数 α。

(2) 根据公式(4.1)生成两个新个体。

遗传算法的收敛性主要取决于其核心操作交叉算子的收敛性。由交叉算子的搜索能力，可以得出结论：只在交叉算子的作用下，随着演化代数的增加，模式内部的各基因将趋于独立，并且只要组成模式的各基因都存在，则该模式一定能被搜索到，此时模式的极限概率等于组成该模式各基因的初始概率(也就是基因的极限概率)的乘积，并且与模式的定义距无关，从而说明了交叉算子能使群体分布扩充的特性。

## 4.4 变　　异

在生物的遗传和自然进化过程中，其细胞分裂复制环节有可能会因为某些偶然因素的影响而产生一些复制差错，这样会导致生物的某些基因发生某种变异，从而产生出新的染色体，表现出新的生物性状。遗传算法模仿生物遗传和进化过程中的变异环节。变异(Mutation)是以较小的概率对个体编码串上的某个或某些位值进行改变，如二进制编码中“0”变为“1”，“1”变为“0”，进而生成新个体。在遗传算法中也引入了变异算子来产生新的个体。遗传算法中所谓的变异运算，是指将个体染色体编码串中的某些基因座上的基因值用该基因座的其他等位基因来替换，从而形成一个新的个体。从遗传运算过程中产生新个体的能力方面来说，变异本身是一种随机算法，但与选择和交叉算子结合后，能够避免由于选择和交叉运算而造成的某些信息丢失，保证遗传算法的有效性。交叉运算是产生新个体的主要方法，它决定了遗传算法的全局搜索能力；而变异运算只是产生新个体的辅助方法，但它也是必不可少的一个步骤，因为它决定了遗传算法的局部搜索能力。交叉算子与变异算子相互配合，共同完成对搜索空间的全局搜索和局部搜索，从而使得遗传算法能够以良好的搜索性能完成最优化问题的寻优过程。

在遗传算法中使用变异算子主要有以下两个目的：

(1) 改善遗传算法的局部搜索能力。遗传算法使用交叉算子已经从全局的角度出发找

到了一些较好的个体编码结构，它们已接近或有助于接近问题最优解。但仅使用交叉算子无法对搜索空间的细节进行局部搜索。这时若再使用变异算子来调整个体编码串中的部分基因值，就可以从局部的角度出发使个体更加逼近最优解，从而提高了遗传算法的局部搜索能力。

(2) 维持群体的多样性，防止出现早熟现象。变异算子用新的基因值替换原有基因值，从而可以改变个体编码串的结构，维持群体的多样性，这样有利于防止出现早熟现象。变异算子使得遗传算法在接近最优解邻域时能加速向最优解收敛，并可以维持群体多样性，避免未成熟收敛。

表 4.5 所示为变异算子。

表 **4.5** 变异算子

| 序号 | 名称 | 特点 | 适用编码 |
| --- | --- | --- | --- |
| 1 | 基本位突变 | 标准遗传算法成员 | 符号 |
| 2 | 有效基因突变 | 避免有效基因缺失 | 符号 |
| 3 | 自适应有效基因突变 | 最低有效基因个数自适应变化 | 符号 |
| 4 | 概率自调整突变 | 由两个串的相似性确定突变概率 | 符号 |
| 5 | 均匀突变 | 每一个实数元素以相同的概率在域内变动 | 实数 |
| 6 | 非均匀突变 | 使整个矢量在解空间轻微变动 | 实数 |
| 7 | 边界变异 | 适用于最优点位于或接近于可行解的边界时的一类带约束条件问题 | 实数 |
| 8 | 高斯近似突变 | 提高对重点搜索区域的局部搜索性能力 | 实数 |
| 9 | 零突变 | | 实数 |

下面介绍几种变异操作方法，它们适合于二进制编码的个体和浮点数编码的个体。

**1.** 基本位变异

基本位变异(Simple Mutation)操作是指对个体编码串中以变异概率、随机指定的某一位或某几位基因座上的值做变异运算，其具体操作过程如下：

(1) 对个体的每一个基因座，以变异概率指定其为变异点。

(2) 对每一个指定的变异点，对其基因值做取反运算或用其他等位基因值来代替，从而产生出新一代的个体。

**2.** 均匀变异

均匀变异(Uniform Mutation)操作是指分别用符合某一范围内均匀分布的随机数，以某一较小的概率来替换个体编码串中各个基因座上的原有基因值。

均匀变异的具体操作过程是：

(1) 依次指定个体编码串中的每个基因座为变异点。

(2) 对每一个变异点，以变异概率从对应基因的取值范围内取一随机数来替代原有值。

均匀变异操作特别适合应用于遗传算法的初级运行阶段，它使得搜索点可以在整个搜索空间内自由地移动，从而可以增加群体的多样性，使算法处理更多的模式。

**3. 边界变异**

边界变异(Boundary Mutation)操作是上述均匀变异操作的一个变形遗传算法。在进行边界变异操作时，随机地取基因座的两个对应边界基因值之一去替代原有基因值。当变量的取值范围特别宽，并且无其他约束条件时，边界变异会带来不好的作用，但它特别适用于最优点位于或接近于可行解的边界时的一类问题。

**4. 非均匀变异**

均匀变异操作取某一范围内均匀分布的随机数来替换原有基因值，可使得个体在搜索空间内自由移动，但另一方面，它却不便于对某一重点区域进行局部搜索。为改进这个性能，我们不是取均匀分布的随机数去替换原有的基因值，而是对原有的基因值做一随机扰动，以扰动后的结果作为变异后的新基因值。对每个基因座都以相同的概率进行变异运算之后，相当于整个解向量在解空间中作了一个轻微的变动。这种变异操作方法就称为非均匀变异(Non-Uniform Mutation)。

**5. 高斯近似变异**

高斯变异(Gaussian Mutation)是改进遗传算法对重点搜索区域的局部搜索性能的另一种变异操作方法。所谓高斯变异操作，是指进行变异操作时用符合均值为 $\bar{P}$、方差为 $P^2$ 的正态分布的一个随机数来替换原有的基因值。由正态分布的特性可知，高斯变异也是重点搜索原个体附近某个局部区域。高斯变异的具体操作过程与均匀变异相类似。

## 4.5 适应度函数

在遗传算法中使用适应度(Fitness)这个概念来度量群体中各个个体在优化计算中能达到或接近于或有助于找到最优解的优良程度。适应度较高的个体遗传到下一代的概率就较大；而适应度较低的个体遗传到下一代的概率就相对小一些。度量个体适应度的函数称为适应度函数(Fitness Function)。

适应度函数也称为评价函数，是根据目标函数确定的用于区分群体中个体好坏的标准，是算法演化过程的驱动力，也是进行自然选择的惟一依据。适应度函数总是非负的，任何情况下都希望其值越大越好。而目标函数可能有正有负，即有时求最大值，有时求最小值，因此需要在目标函数与适应度函数之间进行变换。为了变更选择压力，也需要对适应度函数进行变换。

评价个体适应度的一般过程为：

(1) 对个体编码串进行解码处理后，可得到个体的表现型。

(2) 由个体的表现型可计算出对应个体的目标函数值。

(3) 根据最优化问题的类型，由目标函数值按一定的转换规则求出个体的适应度。

### 4.5.1 适应度函数的作用

在选择操作时会出现以下两个称为遗传算法的欺骗问题。

(1) 在遗传进化的初期，通常会产生一些超常的个体，若按照比例选择法，这些超常个体会因竞争力突出而控制选择过程，影响算法的全局优化性能。

(2) 在遗传进化的后期，即算法接近收敛时，由于种群中个体适应度差异较小时，继

续优化的潜能降低，可能获得某个局部最优解。如果适应度函数设计不当，有可能造成这两种问题的出现。所以适应度函数的设计是遗传算法设计的一个重要方面。

### 4.5.2 适应度函数的设计主要满足的条件

适应度函数的设计主要应满足以下条件：

（1）单值、连续、非负、最大化。这个条件很容易理解和实现。

（2）合理、一致性。要求适应度值反映对应解的优劣程度，这个条件的达成往往比较难以衡量。

（3）计算量小。适应度函数设计应尽可能简单，这样可以减少计算时间和空间上的复杂性，降低计算成本。

（4）通用性强。适应度对某类具体问题，应尽可能通用，最好无需使用者改变适应度函数中的参数。

### 4.5.3 适应度函数的种类

由解空间中某一点的目标函数值 $f(x)$ 到搜索空间中对应个体的适应度函数值 $Fit(f(x))$ 的转换方法基本上有以下三种：

（1）直接以待解的目标函数 $f(x)$ 转化为适应度函数 $Fit(f(x))$，令

$$Fit(f(x)) = \begin{cases} f(x) & \text{目标函数为最大化问题} \\ -f(x) & \text{目标函数为最小化问题} \end{cases} \tag{4.2}$$

这种适应度函数简单直观，但存在两个问题：一是可能不满足常用的轮盘赌选择中概率非负的要求；二是某些待求解的函数在函数值分布上相差很大，由此得到的平均适应度可能不利于体现种群的平均性能，而影响算法的性能。

（2）对于求最小值的问题，做下列转换：

$$Fit(f(t)) = \begin{cases} c_{\max} - f(x) & f(x) < c_{\max} \\ 0 & \text{其他} \end{cases} \tag{4.3}$$

式中，$c_{\max}$ 为一个适当的相对比较大的数，是 $f(x)$ 的最大值估计，可以是一个合适的输入值。

对于求最大值的问题，做下列转换：

$$Fit(f(t)) = \begin{cases} f(x) + c_{\min} & f(x) > c_{\min} \\ 0 & \text{其他} \end{cases} \tag{4.4}$$

式中，$c_{\min}$ 为 $f(x)$ 的最小值估计，可以是一个合适的输入值。

这种方法是第一种方法的改进，可以称为“界限构造法”，但这种方法有时存在界限值预先估计困难、不可能精确的问题。

（3）若目标函数为最小值问题，令

$$Fit(f(t)) = \frac{1}{1 + c + f(x)}, \quad c \geqslant 0，c + f(x) \geqslant 0 \tag{4.5}$$

若目标函数为最大值问题，令

$$Fit(f(t)) = \frac{1}{1 + c - f(x)}, \quad c \geqslant 0，c - f(x) \geqslant 0 \tag{4.6}$$

这种方法与第二种方法类似，$c$ 为目标函数界限的保守估计值。

### 4.5.4 适应度尺度的变换

在遗传算法中，各个个体被遗传到下一代的群体中的概率是由该个体的适应度来确定的。应用实践表明，如何确定适应度对遗传算法的性能有较大的影响。有时在遗传算法运行的不同阶段，还需要对个体的适应度进行适当的扩大或缩小。这种对个体适应度所做的扩大或缩小变换就称为适应度尺度变换(Fitness Scaling Transform)。

常用的尺度变换方法有三种：线性尺度变换、乘幂尺度变换和指数尺度变换。

**1. 线性尺度变换**

线性尺度变换的公式为

$$F' = aF + b \tag{4.7}$$

式中，$F$ 为原适应度；$F'$ 为尺度变换后的新适应度；$a$ 和 $b$ 为系数。$a$ 和 $b$ 的确定直接影响到这个尺度变换的大小，所以对其选取有一定的要求，要满足两个条件：

(1) 原适应度的平均值 $F_{avg}$ 要等于定标后的适应度平均值 $F'_{avg}$，以保证适应度为平均值的个体在下一代的期望复制数为 1，即 $F'_{avg}=F_{avg}$。

(2) 变换后的适应度最大值应等于原适应度平均值的指定倍数，以控制适应度最大的个体在下一代中的复制数，即

$$F'_{avg} = C \cdot F_{avg} \tag{4.8}$$

式中，$C$ 为最佳个体的期望复制数量。试验表明，指定倍数可在 1.0～2.0 范围内。对于群体规模大小为 50～100 个个体的情况，一般取 $C=1.2\sim2$。这个条件是为了保证群体中最好的个体能够期望复制 $C$ 倍到新一代群体中。系数 $a$、$b$ 可按线性比例确定：

$$\begin{cases} a = \dfrac{(C-1)\cdot F_{avg}}{F_{max}-F_{avg}} \\ b = \dfrac{(F_{max}-C\cdot F_{avg})\cdot F_{avg}}{F_{max}-F_{avg}} \end{cases} \tag{4.9}$$

使用线性尺度变换时，变换了适应度之间的差距，保持了种群内的多样性，并且计算简单，易于实现。如图 4.6 所示，群体中少数几个优良个体的适应度按比例缩小，同时几个较差个体的适应度也按比例扩大。

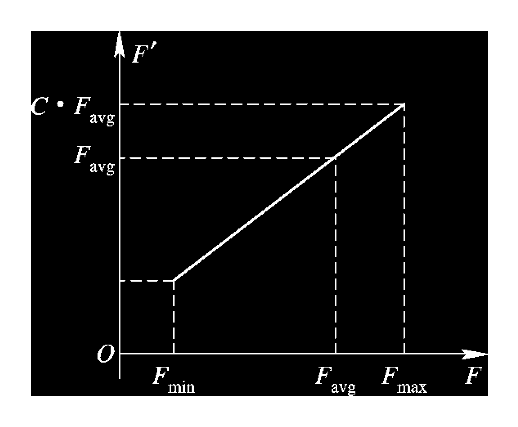



图 4.6　线性尺度变换的正常情况

如果种群内某些个体适应度远低于平均值时，有可能出现变换后适应度值为负的情况，如图 4.7 所示。考虑到要保证最小适应度非负的条件，进行如下的变换：

$$\begin{cases} a = \dfrac{F_{avg}}{F_{avg} - F_{min}} \\ b = \dfrac{-F_{min} \cdot F_{avg}}{F_{avg} - F_{min}} \end{cases} \tag{4.10}$$

图 4.8 所示为利用式(4.10)变换后的结果。

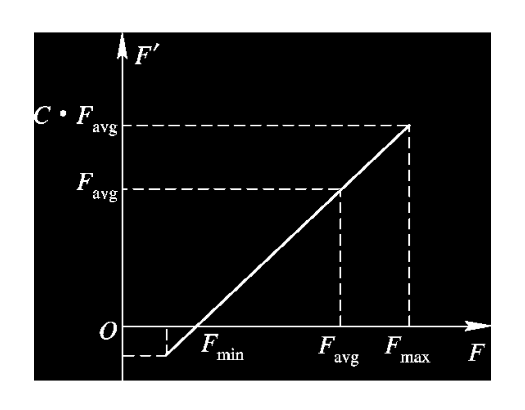



图 4.7 线性尺度变换的异常情况

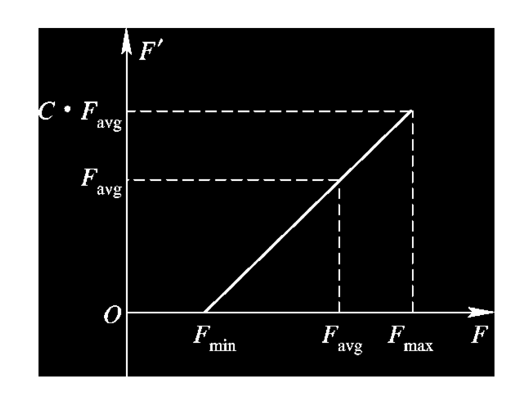



图 4.8 适应度出现负值时的调整

**2.** 乘幂尺度变换

乘幂尺度变换时新的适应度是原适应度的某个指定乘幂。乘幂尺度变换的公式为

$$F' = F^k \tag{4.11}$$

式中,幂指数 $k$ 与所求解的问题有关,且在算法的执行过程中需要不断对其进行修正才能使尺度变换满足一定的伸缩要求。

**3.** 指数尺度变换

指数尺度变换时新的适应度是原适应度的某个指数。指数尺度变换的公式为

$$F' = \exp(-\beta F) \tag{4.12}$$

式中,系数 $\beta$ 决定了选择的强制性,$\beta$ 越小,原有适应度较高的个体的新适应度就越与其他个体的新适应度相差较大,即越增加了选择该个体的强制性。

## 4.6 控制参数选择

遗传算法中的控制参数选择非常关键,控制参数的不同选取会对遗传算法的性能产生较大的影响,影响到整个算法的收敛性。这些参数包括群体规模 $N$、二进制(十进制)编码长度、交叉概率 $P_c$、变异概率 $P_m$ 等。简单遗传算法对其中的参数选择比较敏感。

优化过程中,交叉概率始终控制着遗传运算中起主导地位的交叉算子。不适合的交叉概率会导致意想不到的后果。交叉概率控制着交叉操作被使用的频度。较大的交叉概率可使各代充分交叉,但群体中的优良模式遭到破坏的可能性增大,以致产生较大的代沟,从而使搜索走向随机化;交叉概率越低,产生的代沟就越小,这样将保持一个连续的解空间,使找到全局最优解的可能性增大,但进化的速度就越慢;若交叉概率太低,就会使得更多的个体直接复制到下一代,遗传搜索可能陷入停滞状态。一般建议 $P_c$ 取值范围是 0.4～0.99。变异运算是对遗传算法的改进,对交叉过程中可能丢失的某种遗传基因进行修复和补充,也可防止遗传算法尽快收敛到局部最优解。变异概率控制着变异操作被使用的频度。变异概率取值较大时,虽然能够产生较多的个体,增加了群体的多样性,但也有可能

破坏掉很多好的模式，使得遗传算法的性能近似于随机搜索算法的性能；若变异概率取值太小，则变异操作产生新个体和抑制早熟现象的能力就会较差。实际应用中发现：当变异概率 $P_m$ 很小时，解群体的稳定性好，一旦陷入局部极值就很难跳出来，易产生未成熟收敛；而增大 $P_m$ 的值(如 0.08)，可破坏解群体的同化，使解空间保持多样性，搜索过程可以从局部极值点跳出来，收敛到全局最优解。在求解过程中也可以使用可变的 $P_m$，即算法早期 $P_m$ 取值较大，扩大搜索空间；算法后期 $P_m$ 取值较小，加快收敛速度。一般建议的取值范围是 0.0001～0.1。交叉运算是产生新个体的主要方法，它决定了遗传算法的全局搜索能力，而变异操作只是产生新个体的辅助方法。但它决定了遗传算法的局部搜索能力。交叉算子和变异算子相互配合，共同完成对搜索空间的全局搜索和局部搜索，从而使得遗传算法能够以良好的搜索性能完成最优问题的寻优过程。

群体规模(Population)的大小直接影响到遗传算法的收敛性或计算效率。规模过小，容易收敛到局部最优解；规模过大，会造成计算速度降低。群体规模可以根据实际情况在 10～200 之间选定。

## 4.7 约束条件的处理

在遗传算法中必须对约束条件(Constraints)进行处理，但目前尚无处理各种约束条件的一般方法，根据具体问题可选择下列三种方法，即搜索空间限定法、可行解变换法和罚函数法。

**1.** 搜索空间限定法

搜索空间限定法的基本思想是对遗传算法的搜索空间的大小加以限制，使得搜索空间中表示一个个体的点与解空间中的表示一个可行解的点有一一对应的关系。对一些比较简单的约束条件通过适当编码使搜索空间与解空间一一对应，限定搜索空间能够提高遗传算法的效率。在使用搜索空间限定法时必须保证交叉、变异之后的新个体在解空间中有对应解。

**2.** 可行解变换法

可行解变换法的基本思想是在由个体基因型到个体表现型的变换中，增加使其满足约束条件的处理过程，即寻找个体基因型与个体表现型的多对一变换关系，扩大了搜索空间，使进化过程中所产生的个体总能通过这个变换而转化成解空间中满足约束条件的一个可行解。可行解变换法对个体的编码方法、交叉运算、变异运算等无特殊要求，但运行效率下降。

**3.** 罚函数法

罚函数法的基本思想是对在解空间中无对应可行解的个体计算其适应度时，处以一个罚函数，从而降低该个体的适应度，使该个体被遗传到下一代群体中的概率减小。可以用下式对个体的适应度进行调整：

$$F'(x) = \begin{cases} F(x) & x\text{满足约束条件} \\ F(x) - P(x) & x\text{不满足约束条件} \end{cases} \tag{4.13}$$

式中，$F(x)$为原适应度；$F'(x)$为调整后的新适应度；$P(x)$为罚函数。

如何确定合理的罚函数是这种处理方法难点之所在，在考虑罚函数时，既要度量解对约束条件不满足的程度，又要考虑计算效率。

# 第五章　遗传算法工具箱函数

本章将详细介绍英国设菲尔德大学开发的遗传算法工具箱函数。

由于 MATLAB 高级语言的通用性，对问题用 M 文件编码，与此配对的是 MATLAB 先进的数据分析、可视化工具、特殊目的的应用领域工具箱和展现给使用者具有研究遗传算法可能性的一致环境。MATLAB 遗传算法工具箱为遗传算法从业者和初次实验遗传算法的人提供了广泛多样的有用函数。

遗传算法工具箱使用 MATLAB 矩阵函数为实现广泛领域的遗传算法建立了一套通用工具，这个遗传算法工具是用 M 文件写成的命令行形式的函数，是完成遗传算法大部分重要功能的程序的集合。用户可通过这些命令行函数，根据实际分析的需要，编写出功能强大的 MATLAB 程序。

## 5.1　工具箱结构

本节将给出 GA 工具箱的主要程序。表 5.1 所示为遗传算法工具箱中的函数分类表。

表 **5.1**　遗传算法工具箱中的函数分类表

| | 函　　数 | 功　　能 |
|---|---|---|
| 创建种群 | crtbase | 创建基向量 |
| | crtbp | 创建任意离散随机种群 |
| | crtrp | 创建实值初始种群 |
| 适应度计算 | ranking | 常用的基于秩的适应度计算 |
| | scaling | 比率适应度计算 |
| 选择函数 | reins | 一致随机和基于适应度的重插入 |
| | rws | 轮盘选择 |
| | select | 高级选择例程 |
| | sus | 随机遍历采样 |
| 变异算子 | mut | 离散变异 |
| | mutate | 高级变异函数 |
| | mutbga | 实值变异 |

| | 函　　数 | 功　　能 |
|---|---|---|
| 交叉算子 | recdis | 离散重组 |
| | recint | 中间重组 |
| | reclin | 线性重组 |
| | recmut | 具有变异特征的线性重组 |
| | recombin | 高级重组算子 |
| | xovdp | 两点交叉算子 |
| | xovdprs | 减少代理的两点交叉 |
| | xovmp | 通常多点交叉 |
| | xovsh | 洗牌交叉 |
| | xovshrs | 减少代理的洗牌交叉 |
| | xovsp | 单点交叉 |
| | xovsprs | 减少代理的单点交叉 |
| 子种群的支持 | migrate | 在子种群间交换个体 |
| 实用函数 | bs2rv | 二进制串到实值的转换 |
| | rep | 矩阵的复制 |

### 5.1.1 种群表示和初始化

种群表示和初始化函数有：crtbase，crtbp，crtrp。

GA 工具箱支持二进制、整数和浮点数的基因表示。二进制和整数种群可以使用工具箱中的 crtbp 建立二进制种群。crtbase 是附加的功能，它提供向量描述整数表示。种群的实值可用 crtrp 进行初始化。在二进制代码和实值之间的变换可使用函数 bs2rv，它支持格雷码和对数编码。

### 5.1.2 适应度计算

适应度函数有：ranking，scaling。

适应度函数用于转换目标函数值，给每一个个体一个非负的价值数。这个工具箱支持 Goldberg 的偏移法(Offsetting)和比率法以及贝克的线性评估算法。另外，ranking 函数支持非线性评估。

### 5.1.3 选择函数

选择函数有：reins，rws，select，sus。

这些函数根据个体的适应度大小在已知种群中选择一定数量的个体，对它的索引返回一个列向量。现在最合适的是轮盘赌选择(即 rws 函数)和随机遍历抽样(即 sus 函数)。高级入口函数 select 为选择程序，特别为多种群的使用提供了一个方便的接口界面。在这种

情况下，代沟是必需的，即整个种群在每一代中没有被完全复制。reins 能使用均匀的随机数或基于适应度的重新插入。

### 5.1.4 交叉算子

交叉算子函数有：recdis，recint，reclin，recmut，recombin，xovdp，xovdprs，xovmp，xovsh，xovshrs，xovsp，xovsprs。

交叉是通过给定的概率重组一对个体而产生后代的。单点交叉、两点交叉和洗牌交叉是由 xovsp、xovdp 和 xovsh 函数分别完成的。缩小代理交叉函数分别是：xovdprs、xovshrs 和 xovsprs。通用的多点交叉函数是 xovmp，它提供均匀交换的支持。为支持染色体实值表示，离散的、中间的和线性重组分别由函数 recdis、recint、reclin 完成。函数 recmut 提供具有突变特征的线性重组。函数 recombin 是一高级入口函数，对所有交叉操作提供多子群支持入口。

### 5.1.5 变异算子

变异算子函数有：mut，mutate，mutbga。

二进制和整数变异操作由 mut 完成。实值的变异使用育种机函数 mutbga 是有效的。mutate 对变异操作提供一个高级接口。

### 5.1.6 多子群支持

多子群支持函数：migrate。

遗传算法工具箱通过高层遗传操作函数 migrate 对多子群提供支持，它的一个功能是在子群中交换个体。一个单一种群通过使用工具箱中的函数修改数据结构，使其分为许多子种群，这些子种群被保存在连续的数据单元块中。高层函数(如 select 和 reins)可独立地操作子种群，包含在一个数据结构中的每一子种群允许独自向前衍化。基于孤岛或回迁模式，migrate 允许个体在子种群中迁移。

## 5.2 遗传算法中的通用函数

在这一节中，将详细介绍在 MATLAB 中用于遗传算法的各种工具箱函数，对每个函数从功能、语法格式、使用说明以及用法举例等方面进行阐述。关于每个函数的适用信息由在线帮助工具提供。

### 5.2.1 函数 bs2rv

功能：二进制串到实值的转换。

格式：Phen = bs2rv(Chrom，FieldD)

详细说明：Phen = bs2rv(Chrom，FieldD)根据译码矩阵 FieldD 将二进制串矩阵 Chrom 转换为实值向量。返回矩阵 Phen 包含对应的种群表现型。

使用格雷编码的二进制染色体表示被推荐作为量化间隔的规则海明距离，可使遗传搜索减少欺骗。设置量化点间刻度的可选方案是选择线性或对数编码从二进制变换到实值。

对数刻度用于决策变量的范围未知，作为大范围参数的边界时，搜索可用较少的位数，以减少 GA 的内存需求和计算量。

矩阵 ***FieldD*** 有如下结构：

$$\boldsymbol{FieldD} = \begin{bmatrix} len \\ lb \\ ub \\ code \\ scale \\ lbin \\ ubin \end{bmatrix}$$

这里矩阵的行组成如下：

*len* 是包含在 *Chrom* 中的每个子串的长度，注意 *sum*(*len*)等于 *length*(*Chrom*)。

*lb* 和 *ub* 是行向量，分别指明每个变量使用的下界和上界。

*code* 是二进制行向量，指明子串是怎样编码的。*code*(*i*)＝1 为标准的二进制编码，*code*(*i*)＝0 则为格雷编码。

*scale* 是二进制行向量，指明每个子串是否使用对数或算术刻度。*scale*(*i*)＝0 为算术刻度，*scale*(*i*)＝1 则为对数刻度。

*lbin* 和 *ubin* 是二进制行向量，指明表示范围中是否包含每个边界。选择 *lbin*＝0 或 *ubin*＝0，从表示范围中去掉边界；*lbin* ＝1 或 *ubin* ＝ 1 则在表示范围中包含边界。

例 **5.1**　函数 bs2rv 的应用举例。下列二进制种群 Chrom 由函数 crtbp 创建，表示在[－1，10]之间的一组简单变量，程序代码表示怎样使用函数 bs2rv 将算术表示的格雷码或二进制串表示转换为实值表现型。

```
Chrom＝crtbp(4，8)          % 创建任意染色体，如为二进制串
```

$$\text{Chrom}=\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \end{bmatrix}$$

```
FieldD＝[8；－1；10；1；0；1；1]；        % 包括边界
Phen＝bs2rv(Chrom，FieldD)               % 转换二进制到实值，使用算术刻度
```

$$\text{Phen}=\begin{bmatrix} -0.7843 \\ 9.3961 \\ 1.0706 \\ 5.2980 \end{bmatrix}$$

```
FieldD＝[8；1；10；1；1；0；0]；          % 不包括边界
Phen＝bs2rv(Chrom，FieldD)               % 转换二进制到实值，使用对数刻度
```

$$\text{Phen}=\begin{bmatrix} 6.6223 \\ 5.0615 \\ 2.7277 \\ 1.5236 \end{bmatrix}$$

算法说明：bs2rv 作为 GA 工具箱的一个 M 文件执行，如果使用对数刻度，其范围不能包含零。

### 5.2.2 函数 crtbase

功能：创建基向量。

格式：BaseVec=crtbase(Lind，Base)

详细说明：crtbase 产生向量的元素对应染色体结构的基因座，使用不同的基本字符表示建立种群时这个函数可与函数 crtbp 联合使用。

BaseVec=crtbase(Lind，Base)创建长度为 Lind 的向量，它的每个元素由基本字符决定，如果 Lind 是向量，BaseVec 的长度为 Lind 的总长，如果 Base 也是一个长为 Lind 的向量，则 BaseVec 是一组由 Lind 和基本字符 Base 的元素决定长度的基本字符组组成。当描述染色体结构的基因位基本字符时，最后一选项是有用的。

例 **5.2** 函数 crtbase 的应用举例。下面的程序代码将为种群创建一有 4 个基数为 8 的基本字符{0，1，2，3，4，5，6，7}和 6 个基数为 5 的基本字符{0，1，2，3，4}的基本字符向量。

```
BaseV=crtbase ([4  6],[8  5])
BaseV=[8  8  8  8  5  5  5  5  5  5]
```

参见：crtbp，bs2rv。

### 5.2.3 函数 crtbp

功能：创建初始种群。

格式：① [Chrom，Lind，BaseV] = crtbp(Nind，Lind)

② [Chrom，Lind，BaseV] = crtbp(Nind，BaseV)

③ [Chrom，Lind，BaseV] = crtbp(Nind，Lind，Base)

详细说明：遗传算法的第一步是创建由任意染色体组成的原始种群。crtbp 创建一元素为随机数的矩阵 Chrom。

格式①创建一大小为 Nind×Lind 的随机二元矩阵，这里 Nind 指定种群中个体的数量，Lind 指定个体的长度。此格式习惯于指定染色体的尺寸(维度)。

格式②返回长度为 Lind 的染色体结构，染色体基因位的基本字符由向量 BaseV 决定。

格式③用于产生基本字符为 Base 的染色体矩阵。如果 Base 是向量，则 Base 的元素值指定了染色体的基因位的基本字符。在这种情况下，右边的第二个变元可省略，即为格式②。

例 **5.3** 使用函数 crtbp 创建初始种群的应用举例。

(1) 创建一个长度为 9、有 6 个个体的随机种群：

```
[Chrom，Lind，BaseV] = crtbp (6，9)
```

或

```
[Chrom，Lind，BaseV] = crtbp(6，9，BaseV)
```

运行后得

$$
\text{Chrom}=\begin{bmatrix}
0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\
0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1
\end{bmatrix}
$$

Lind = 9；

BaseV = [ 2 2 2 2 2 2 2 2 2 ]；

(2) 创建一长度为 9、有 6 个个体的随机种群(这里前四个基因位是基本字符{0，1，2，3，4，5，6，7}，后五个基因位是基本字符{0，1，2，3})：

BaseV=crtbase([4 5]，[8 4])；

[Chrom，Lind，BaseV] = crtbp(6，BaseV)；

或

[Chrom，Lind，BaseV]=crtbp([6，9]，[8 8 8 8 4 4 4 4 4])；

运行后得

$$
\text{Chrom}=\begin{bmatrix}
4 & 3 & 1 & 1 & 2 & 0 & 2 & 0 & 3 \\
1 & 4 & 7 & 5 & 2 & 1 & 1 & 1 & 0 \\
1 & 3 & 0 & 1 & 0 & 0 & 0 & 0 & 2 \\
1 & 5 & 5 & 7 & 2 & 0 & 2 & 3 & 1 \\
4 & 5 & 7 & 7 & 0 & 1 & 3 & 0 & 3 \\
4 & 2 & 4 & 0 & 3 & 3 & 1 & 1 & 0
\end{bmatrix}
$$

Lind=9；

BaseV=[8 8 8 8 4 4 4 4 4]；

算法说明：crtbp 是 GA 工具箱中的一个 M 文件，它使用了 MATLAB 随机函数 rand。

参见：crtbase，crtrp。

### 5.2.4 函数 crtrp

功能：创建实值原始种群。

遗传算法的第一步是创建由任意个体组成的原始种群。crtrp 创建元素为均匀分布随机数的矩阵。

格式：Chrom = crtrp(Nind，FieldDR)

详细说明：Chrom = crtrp(Nind，FieldDR)创建一大小为 Nind×Nvar 的随机实值矩阵，这里 Nind 指定了种群中个体的数量，Nvar 指定每个个体的变量个数，Nvar 来自 FieldDR，Nvar=size(FieldDR，2)。

FieldDR (FieldDescriptionRealvalue)是一大小为 2×Nvar 的矩阵，并包含每个个体变量的边界，第一行包含下界，第二行包含上界。

FieldDR 被用在另一些函数中(变异)。

例 **5.4** 使用函数 crtrp 创建一具有 6 个个体，每个个体有 4 个变量的随机种群。

定义边界变量：

$$
FieldDR=\begin{bmatrix} -100 & -50 & -30 & -20 \\ 100 & 50 & 30 & 20 \end{bmatrix} \begin{matrix} \%下界 \\ \%上界 \end{matrix}
$$

创建初始种群：

Chrom＝crtrp(6，FieldDR)

$$
Chrom=\begin{bmatrix} 40.23 & -17.17 & 28.95 & 15.38 \\ 82.06 & 13.26 & 13.35 & -9.09 \\ 52.43 & 25.64 & 15.20 & -2.54 \\ -47.50 & 49.10 & 9.09 & 10.65 \\ -90.50 & -13.46 & -25.63 & -0.89 \\ 47.21 & -25.29 & 7.89 & -10.48 \end{bmatrix}
$$

参见：mutbga，recdis，recint，reclin。

### 5.2.5 函数 migrate

功能：在子种群间迁移个体。

格式：① Chrom = migrate(Chrom，SUBPOP)

② Chrom = migrate(Chrom，SUBPOP，MigOpt)

③ Chrom = migrate(Chrom，SUBPOP，MigOpt，ObjV)

④ [Chrom，ObjV] = migrate(Chrom，SUBPOP，MigOpt，ObjV)

详细说明：migrate 完成在当前种群 Chrom 的子种群间迁移个体，并返回迁移后的种群 Chrom。Chrom 的一行对应一个个体。子种群的数量由 SUBPOP 指定。在 Chrom 中的子种群按照下列方案进行有序排列：

$$
Chrom=\begin{bmatrix} IND_1\,SubPop_1 \\ IND_2\,SubPop_1 \\ \vdots \\ IND_N\,SubPop_1 \\ IND_1\,SubPop_2 \\ IND_2\,SubPop_2 \\ \vdots \\ IND_N\,SubPop_2 \\ \vdots \\ IND_1\,SubPop_{SUBPOP} \\ IND_2\,SubPop_{SUBPOP} \\ \vdots \\ IND_N\,SubPop_{SUBPOP} \end{bmatrix}
$$

所有子种群必须有相同数量的个体。

MigOpt 是一最大具有三个参数的可选向量。

MigOpt(1)是个体在子种群间的迁移概率。如果省略或为 NaN，则假定 MigOpt(1) = 0.2（即 20%）。如果迁移概率不为 0，则每个子种群至少有一个个体迁移。

MigOpt(2)是指定迁移选择方式的标量。0 为均匀迁移，1 为基于适应度的迁移。如果省略或为 NaN，则假设 MigOpt(2)=0。

MigOpt(3)是指定迁移种群结构的标量。0 为完全网状结构，1 为近邻结构，2 为环状结构。如果省略或为 NaN，则假设 MigOpt(3)=0。

如果省略 MigOpt 或为 NaN，则假设为缺省值。

ObjV 是具有与 Chrom 同样行数的任选列向量，并包含 Chrom 所有个体对应的目标函数值。对于基于适应度的迁移(MigOpt(2)=1)，ObjV 为必选项。如果 ObjV 是一输入/输出参数，则目标函数值将按照迁移个体进行拷贝，并保存重新计算的整个种群的目标函数值。

例 **5.5**　列举函数 migrate 各种调用的情形。

Chrom=(Chrom，SUBPOP)在一子种群中选择 20%的个体，并用均匀选择方式与从其他所有子种群中选择的个体进行替换。这个过程将对所有子种群做一遍。其中，MigOpt=[0.2，0，0]。

Chrom=migrate(Chrom，SUBPOP，[NaN 1 NaN]，ObjV)在一子种群中选择 20%的个体并用从其他所有子种群中选择最适应(较小的 ObjV)的个体替换。(网状结构)这个过程将对所有子种群重复。

[Chrom，ObjV]=migrate(Chrom，SUBPOP，[0.3 1 2]，ObjV)在一子种群中选择 30%的个体并用从单向环状结构的邻近子种群中选择最适应(较小的 ObjV)的个体替换。这个过程将对所有子种群重复。第一个子种群从最后一个子种群接收它的新的个体(SUBPOP)。ObjV 按照迁移个体进行返回。这个迁移方案如下：

subpop1→subpop2→subpop3→…→subpopSUBPOP→subpop1

[Chrom，ObjV]=migrate(Chrom，SUBPOP，[NaN NaN 1]，ObjV)在一子种群中选择 20%的个体并用均匀选择方式与从两个相邻子种群中选择的个体替换。这个过程将对所有子种群重复一遍。第一个子种群从第二个子种群和最后一个子种群中接收新的个体。最后一个子种群从第一个子种群和第 SUBPOP-1 个子种群接收新的个体。ObjV 按照迁移个体进行返回。这个迁移方案如下：

subpopSUBPOP→subpop1↔subpop2↔…↔subpopSUBPOP↔subpop1

参见：select，recombin，mutate，reins。

### 5.2.6　函数 mut

功能：离散变异算子。

格式：① NewChrom = mut(OldChrom，Pm)

② NewChrom = mut(OldChrom，Pm，BaseV)

详细说明：mut 取当前种群的表示，并用特定概率对每个元素进行变异。如果染色体和种群结构中允许不同的基本字符，则 mut 允许用一个附加的变量 BaseV 来指定染色体中每一个元素的基本字符。

假定种群为二进制编码，NewChrom＝mut(OldChrom，Pm)，取当前种群OldChrom，每一行对应一个体并用概率 Pm 变异每一个元素。如果省略，则假设 Pm＝0.7/Lind，这里 Lind 是染色体结构的长度，这个值的选择将使染色体中的每一个元素的变异概率近似等于 0.5。

NewChrom＝mut (OldChrom，Pm，BaseV)使用第三个变量指明染色体个体元素的变异的基本字符。在这里，length(BaseV)＝Lind，Lind 是染色结构的长度。

mut 是一低级变异函数，通常被 mutate 调用。

例 **5.6** 下面的程序代码为使用函数 mut 将当前种群变异为新种群。

(1) 调用变异函数：

$$\text{OldChrom}=\begin{bmatrix}0&0&0&0&0&1&1&1\\1&0&0&0&1&0&0&1\\0&0&1&0&1&0&0&0\\1&1&0&1&1&0&1&1\end{bmatrix}$$

变异 OldChrom 使用缺省的概率，通过调用变异函数：

NewChrom＝mut(OldChrom)

这时，OldChrom 将变成 NewChrom：

$$\text{NewChrom}=\begin{bmatrix}0&0&1&0&0&1&1&1\\1&1&0&0&0&0&0&1\\0&0&0&0&0&0&0&0\\1&1&0&1&1&0&1&1\end{bmatrix}$$

(2) 创建一长度为 8、有 6 个个体的随机种群：

BaseV ＝[ 8 8 8 4 4 4 4 4];

[Chrom，Lind，BaseV]＝crtbp(6，BaseV)；

$$\text{Chrom}=\begin{bmatrix}6&1&0&3&2&2&2&2\\0&4&0&2&1&1&3&1\\6&0&2&0&2&1&1&3\\5&4&7&3&1&1&1&3\\1&3&7&3&3&2&3&1\\4&1&4&0&0&0&3&3\end{bmatrix}$$

NewChrom＝mut(Chrom，0.0778，BaseV)

因此 Chrom 将变成

$$\text{NewChrom}=\begin{bmatrix}6&1&0&3&2&2&2&2\\0&4&0&2&1&1&3&1\\6&0&2&0&2&2&1&3\\5&4&7&3&1&1&1&3\\1&3&7&3&0&2&3&1\\4&1&4&0&0&0&3&3\end{bmatrix}$$

需要补充说明的是，如果二进制串使用变异概率 1，则调用过程及获得的结果如下：

mut ([1 0 1 0 1 1 1 0]，1)；

运行后得

ans=[0　1　0　1　0　0　0　1]

参见：mutate，mutbga。

### 5.2.7　函数 mutate

功能：个体的变异(高级函数)。

格式：① NewChrom=mutate（MUT_F，OldChrom，FieldDR）

② NewChrom=mutate（MUT_F，OldChrom，FieldDR，MutOpt）

③ NewChrom=mutate（MUT_F，OldChrom，FieldDR，MutOpt，SUBPOP）

详细说明：mutate 执行种群 OldChrom 中个体的变异，并在新种群 NewChrom 中返回变异后的个体，OldChrom 和 NewChrom 中每一行对应一个个体。

MUT_F 是一字符串，包含低级变异函数的名字，例如 mutbga 或 mut。

对实值变量，FieldDR 是一大小为 2×Nvar 的矩阵，指定每个变量的边界；对离散值变量，FieldDR 是一大小为 1×Nvar 的矩阵，指定每个变量的基本字符。如果 FieldDR 省略、空或为 NaN，则假设变量为二进制表示。

MutOpt 是一任选参数项，包含变异概率，即个体变量的突变可能性。如果 MutOpt 省略，则一缺省突变概率被假设。对实值变异，MutOpt 可能包含第二个参数，该参数是说明压缩变异的范围的标量(参见 mutbga)。

SUBPOP 是一任选参数项，决定 OldChrom 中子种群的数量。如果 SUBPOP 省略或为 NaN，则假设 SUBPOP=1。OldChrom 中的所有子种群必须具有相同的大小。

例 **5.7**　变异函数 mutate 的应用举例。下面的程序代码为一个二进制种群和一个十进制种群的变异。

对于二进制种群，选取 MUT_F='mut'。

$$
OldChrom=\begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}
$$

NewChrom=mutate（'mut'，Chrom）；

运行后得

$$
NewChrom=\begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}
$$

对于十进制种群，选取 MUT_F='mutbga'。

Chrom=crtbp(6，BaseV)；　　　% 6 个个体的种群，Base V 同例 5.6

运行后得

$$\text{Chrom}=\begin{bmatrix}2&0&0&2&3&1&2&1\\5&4&4&3&0&1&3&3\\6&6&5&2&2&1&3&2\\3&2&0&2&3&1&3&3\\3&3&7&3&3&3&1&1\\2&0&3&0&2&2&2&3\end{bmatrix}$$

边界定义如下：

$$\text{FieldDR}=\begin{bmatrix}0&0&0&0&0&0&0&0\\8&8&8&4&4&4&4&4\end{bmatrix}$$

NewChrom = mutate ('mutbga', Chrom, FieldDR)； % 将 Chrom 变异为 NewChrom

运行后得

$$\text{NewChrom}=\begin{bmatrix}2&0&0&2&3&1&2&1\\5&4&4&3&0&1&3&3\\6&6&5&2&2&1&3&2\\3&2&0&2&3&1&3&3\\3&3&7&3&3&3&1&1\\2&0&3&0&2&2&2&3\end{bmatrix}$$

算法说明：mutate 检测输入参数的一致性并调用低级变异函数，如果 mutate 被调用于多子群，则每个子群分别调用低级变异函数。

参见：mutbga，mut，recombin，select。

### 5.2.8 函数 mutbga

功能：实值种群的变异(遗传算法育种器的变异算子)。

格式：① NewChrom = mutbga(OldChrom，FieldDR)

② NewChrom = mutbga(OldChrom，FieldDR，MutOpt)

详细说明：mutbga 对实值种群 OldChrom，使用给定的概率，变异每一个变量，返回变异后的种群 NewChrom。

FieldDR 是一矩阵，包含每个变量的边界(参见 crtrp)。

MutOpt 是一可选向量，具有两个参数的最大值。

MutOpt(1)是变异概率。如果缺省或为 NaN，则 MutOpt(1) = 1/Nvar，这里 Nvar 是由 size(FieldDR，2)定义的每个个体的变量数。这个值被选定，则表示每个个体的变异数近似为 1。

MutOpt(2)是[0，1]之间的一个量，压缩变异的范围。如果省略或为 NaN，则假设 MutOpt(2) = 1(不压缩)。

格式②利用保存在矩阵 OldChrom 中的当前种群，使用概率 MutOpt(1)附加的一个小随机值(变异步长大小)变异每个变量，变异步长可由 MutOpt(2)限定。

mutbga 是低级变异函数，通常被 mutate 调用。

例 **5.8** 使用函数 mutbga 变异实值种群的应用举例。考虑以下具有三个实值个体的种群：

$$\text{OldChrom}=\begin{bmatrix}40.2381 & -17.1766 & 28.9530 & 15.3883\\ 82.0642 & 13.2639 & 13.3596 & -9.0916\\ 52.4396 & 25.6410 & 15.2014 & -2.5435\end{bmatrix}$$

边界定义如下：

$$\text{FieldDR}=\begin{bmatrix}-100 & -50 & -30 & -20\\ 100 & 50 & 30 & 20\end{bmatrix}$$

变异 OldChrom 的变异概率为 1/4，不压缩变异范围。

NewChrom = mutbga(OldChrom，FieldDR，[1/4 1.0])；

mutbga 产生一中间任务表 MutMx，决定变异的变量，并为加入的 delta 所标识(参看算法)。例如：

$$\text{MutMx}=\begin{bmatrix}0 & 0 & 0 & 1\\ 0 & 0 & -1 & 0\\ 0 & 0 & -1 & -1\end{bmatrix}$$

第二个中间表 delta 标识正常的变异步长大小。例如：

$$\text{delta}=\begin{bmatrix}0.2500 & 0.2500 & 0.2500 & 0.2500\\ 0.0001 & 0.0001 & 0.0001 & 0.0001\\ 0.2505 & 0.2505 & 0.2505 & 0.2505\end{bmatrix}$$

在变异后，NewChrom 成为

$$\text{NewChrom}=\begin{bmatrix}40.2381 & -17.1766 & 28.9530 & 20.0000\\ 82.0642 & 13.2639 & 13.3559 & -9.0916\\ 52.4396 & 25.6410 & -7.6858 & -7.5539\end{bmatrix}$$

NewChrom－OldChrom 显示变异的步长，即

$$\text{NewChrom}-\text{OldChrom}=\begin{bmatrix}0 & 0 & 0 & 4.6117\\ 0 & 0 & -0.0037 & 0\\ 0 & 0 & -7.5156 & -5.0104\end{bmatrix}$$

算法说明：一个变量的变异是由以下计算得到的：

mutated variable＝variable＋MutMx×range×MutOpt(2)×delta

若具有概率 MutOpt(1)，则 MutMx＝1，否则为 0 (＋或－具有相等概率)。

range＝0.5×变量的域(范围)。

如果 $m=20$，概率为 $1/m$，则 $a_i=1$，否则为 0。

使用 $m=20$，变异算子能定位最优值到精度为 range×MutOpt(2)×$2^{-19}$。

变异算子 mutbga 能在个体变量和变异范围定义的立方体中产生最多的点。这个测试经常接近变量，即小步长概率大于大步长概率。

参见：mutate，recdis，recint，reclin。

### 5.2.9 函数 ranking

功能：基于排序的适应度分配。

格式：① FitnV＝ranking(ObjV)

② FitnV＝ranking(ObjV，RFun)

③ FitnV＝ranking(ObjV，RFun，SUBPOP)

详细说明：ranking 按照个体的目标值 ObjV 由小到大的顺序对它们进行排序，并返回一包含对应个体适应度值 FitnV 的列向量。这个函数是从最小化方向对个体进行排序的。

RFun 是一任选向量，有 1、2 或 length(ObjV)个参数。

如果 RFun 是一在[1，2]内的标量，则采用线性排序，这个标量指定选择的压差。

如果 RFun 是一具有两个参数的向量，则

- RFun(1)：对线性排序，标量指定的选择压差 RFun(1)必须在[1，2]之间；对非线性排序，RFun(1)必须在[1，length(ObjV)－2]之间。如果为 NaN，则 RFun(1)假设为 2。
- RFun(2)：指定排序方法。0 为线性排序，1 为非线性排序。

如果 RFun 为长为 length(ObjV)的向量，则它包含对每一行的适应度值计算。

如果省略 RFun 或为 NaN，则采用线性排序，选择压差假设为 2。

SUBPOP 是一任选参数，指明在 ObjV 中子种群的数量。如果省略 SUBPOP 或为 NaN，则假设 SUBPOP＝1。在 ObjV 中的所有子种群大小必须相同。

如果 ranking 被调用于多子种群，则 ranking 独立地对每个子种群执行。

例 **5.9** 函数 ranking 的应用举例。下面为取不同参数时使用函数 ranking 对 10 个个体的种群进行排序。

考虑具有 10 个个体的种群，其当前目标值如下：

$$\text{ObjV}=[1\quad 2\quad 3\quad 4\quad 5\quad 10\quad 9\quad 8\quad 7\quad 6]^{T}$$

(1) 使用线性排序和选择压差为 2 估算适应度：

FitnV＝ranking(ObjV)

$$\text{FitnV}=\begin{bmatrix}2.0000\\1.7778\\1.5556\\1.3333\\1.1111\\0\\0.2222\\0.4444\\0.6667\\0.8889\end{bmatrix}$$

(2) 使用非线性排序和选择压差为 2 估算适应度：

FitnV＝ranking(ObjV，[2 1])

$$FitnV=\begin{bmatrix}2.0000\\1.6633\\1.3833\\1.1504\\0.9568\\0.3807\\0.4577\\0.5504\\0.6618\\0.7957\end{bmatrix}$$

（3）使用 Rfun 中的值估算适应度：

RFun=[3　5　7　10　14　18　25　30　40　50]

FitnV=ranking(ObjV，RFun)

$$FitnV=\begin{bmatrix}50\\40\\30\\25\\18\\3\\5\\7\\10\\14\end{bmatrix}$$

（4）使用非线性排序，选择压差为 2，在 ObjV 中有两个子种群估算适应度：

$$FitnV=\begin{bmatrix}2.0000\\1.2889\\0.8307\\0.5354\\0.3450\\0.3450\\0.5354\\0.8307\\1.2889\\2.0000\end{bmatrix} \quad 或 \quad FitnV=\begin{bmatrix}2.00\\1.28\\0.83\\0.53\\0.34\\0.34\\0.53\\0.83\\1.28\\2.00\end{bmatrix}$$

FitnV=ranking(ObjV，[2　1]，2)

算法说明：这个算法对线性和非线性排序，它首先对目标函数值进行降序排序。最小适应度个体被放置在排序的目标函数值列表的第一个位置，最适应个体放置在位置 Nind 上。这里 Nind 是种群中个体的数量。每个个体的适应度值根据它在排序种群中的位置 Pos 计算出来。

对线性排序，其适应度值由下式计算：

$$FitnV(Pos) = 2 - sp + 2 \times (sp - 1) \times \frac{Pos - 1}{Nind - 1}$$

对非线性排序：

$$FitnV(Pos) = \frac{Nind \times X^{Pos-1}}{\sum_{i=1}^{Nind} X(i)}$$

这里 $X$ 是多项式方程的方根：

$$0 = (sp - 1) \times X^{Nind-1} + sp \times X^{Nind-2} + \cdots + sp \times X + sp$$

向量 FitnV 是没有排序的，反映原始输入向量 ObjV 的顺序。

参见：select，rws，sus。

### 5.2.10 函数 recdis

功能：离散重组。

格式：NewChrom＝recdis(OldChrom)

详细说明：recdis 完成当前种群 OldChrom 中一对个体的离散重组，在交配后返回新的种群 NewChrom。OldChrom 中每一行对应一个个体。交配的一对是有序的，奇数行与它下一个偶数行配对。如果矩阵 OldChrom 的行数是奇数，最后一个奇数行不交配并添加到 NewChrom 的末端。因此种群被组织成需要交配的连续对。通过使用函数 ranking 达到计算每个个体的适应度水平，由选择函数(例如 select)在当前种群中用与按适应度相关的概率选择个体。

recdis 是一个低级重组函数，通常被函数 recombin 调用。

例 **5.10** 离散重组函数 recdis 的应用举例。下面的程序代码为 5 个实值个体种群的离散重组。

考虑如下具有 5 个实值个体的种群：

$$OldChrom = \begin{bmatrix} 40.23 & -17.17 & 28.95 & 15.38 \\ 82.06 & 13.26 & 13.35 & -9.09 \\ 52.43 & 25.64 & 15.20 & -2.54 \\ -47.50 & 49.10 & 9.09 & 10.65 \\ -90.50 & -13.46 & -25.63 & -0.89 \end{bmatrix}$$

完成离散重组：

NewChrom＝recdis(OldChrom)

recdis 提供一中间掩码表决定哪些父个体为子代贡献哪些变量。例如：

$$Mask = \begin{bmatrix} 1 & 2 & 1 & 2 \\ 2 & 2 & 1 & 1 \\ 2 & 1 & 2 & 1 \\ 1 & 1 & 2 & 2 \end{bmatrix}$$

重组后 NewChrom 成为

$$\text{NewChrom}=\begin{bmatrix} 40.23 & 13.26 & 28.95 & -9.09 \\ 82.06 & 13.26 & 28.95 & 15.38 \\ -47.50 & 25.64 & 9.09 & -2.54 \\ 52.43 & 25.64 & 9.09 & 10.65 \end{bmatrix} \begin{matrix} \text{\% mask(1), 1\&2row} \\ \text{\% mask(2), 1\&2row} \\ \text{\% mask(3), 1\&4row} \\ \text{\% mask(4), 1\&4row} \end{matrix}$$

由于在父种群 OldChrom 的个体数量是奇数，最后一个个体不参加重组而附加到 NewChrom，后代返回到用户空间，因此

$$\text{NewChrom}=\begin{bmatrix} 40.23 & 13.26 & 28.95 & -9.09 \\ 82.06 & 13.26 & 28.95 & 15.38 \\ -47.50 & 25.64 & 9.09 & -2.54 \\ 52.43 & 25.64 & 9.09 & 10.65 \\ -90.50 & -13.46 & -25.63 & -0.89 \end{bmatrix}$$

算法说明：离散重组是在个体间交换变量值，对每个变量，贡献给子代变量值的父亲是随机地以相同的概率挑选的。离散重组产生父母定义的个体的所有可能。

参见：recombin，recint，reclin，ranking，sus，rws。

### 5.2.11 函数 recint

功能：中间重组。

格式：NewChrom＝recint(OldChrom)

详细说明：recint 完成当前种群 OldChrom 成对个体的中间重组，返回交配后的新种群 NewChrom。OldChrom 中每一行对应一个个体。

recint 是一个只能应用于实值(非二进制、非整数)变量种群的函数。

交配的对是有序的，奇行与它下一个偶行配对。如果 OldChrom 的行数是奇数，最后一个奇数行不参与交配，直接加到 NewChrom 的末尾。因此种群根据交配要求组织成连续的对。通过使用 ranking 计算每个个体的适应度水平，由选择函数(例如 select)用与适应度相关的概率在当前种群中选择个体，完成这个重组工作。

recint 是一低级重组函数，通常被 recombin 调用。

例 **5.11** 中间重组函数 recint 的应用举例。下面的程序代码为具有 3 个实值个体的种群的中间重组。

考虑下面具有 3 个实值个体的种群：

$$\text{OldChrom}=\begin{bmatrix} 40.23 & -17.17 & 28.95 & 15.38 \\ 82.06 & 13.26 & 13.35 & -9.09 \\ 52.43 & 25.64 & 15.20 & -2.54 \end{bmatrix} \begin{matrix} \text{\% 父代 1} \\ \text{\% 父代 2} \\ \text{\% 父代 3} \end{matrix}$$

执行中间重组：

NewChrom＝recint(OldChrom)

通过向最先的父个体增加不同微量产生新值，中间比率因子表 Alpha 被产生，例如：

$$\text{Alpha}=\begin{bmatrix} -0.13 & 0.50 & 0.32 & 0.16 \\ 1.12 & 0.54 & 0.44 & 1.16 \end{bmatrix} \begin{matrix} \text{\% 子代 1} \\ \text{\% 子代 2} \end{matrix}$$

因此，重组后的 NewChrom 成为

$$\text{NewChrom}=\begin{bmatrix}34.40 & -1.92 & 23.86 & 11.33\\ 87.11 & -0.59 & 21.98 & -13.04\end{bmatrix}\begin{matrix}\%\text{Alpha(1,:)，父 1 和 2}\\ \%\text{Alpha(2,:)，父 1 和 2}\end{matrix}$$

由于父种群 OldChrom 的个体数是奇数，最后一个个体不参加重组直接加入 NewChrom，因此返回用户空间的后代如下：

$$\text{NewChrom}=\begin{bmatrix}34.40 & -1.92 & 23.86 & 11.33\\ 87.11 & -0.59 & 21.98 & -13.04\\ 52.43 & 25.64 & 15.20 & -2.54\end{bmatrix}$$

算法说明：中间重组重组双亲值使用如下公式：

offspring＝parent1＋Alpha(parent2－parent1)

这里，Alpha 是在区间[－0.25，1.25]内随机一致性选择产生的标量因子。recint 对重组的每一对值产生一新的 Alpha。

中间重组能产生略大于双亲定义的立体空间中的任意点。

中间重组与线性重组 reclin 相似。然而 recint 对每对值使用了一新的 Alpha 值一起重组，而 reclin 对每对双亲使用一 Alpha 因子。

参见：recombin，recdis，reclin，ranking，sus，rws。

### 5.2.12 函数 reclin

功能：线性重组。

格式：NewChrom＝reclin(OldChrom)

详细说明：reclin 完成当前种群 OldChrom 成对个体的线性重组，返回交配后的新种群 NewChrom。OldChrom 中每一行对应一个个体。

reclin 是一个只能应用于实值(非二进制、非整数)变量种群的函数。

交配的对是有序的，奇行与它下一个偶行配对。如果 OldChrom 的行数是奇数，则最后一个奇数行不参与交配，直接加到 NewChrom 的末尾。因此种群根据交配要求组织成连续的对。通过使用 ranking 计算每个个体的适应度水平，由选择函数(例如 select)用与适应度相关的概率在当前种群中选择个体，完成这个重组工作。

reclin 是一低级重组函数，通常被 recombin 调用。

例 **5.12**　函数 reclin 的应用举例。下面的程序代码为具有 3 个实值个体的种群的线性重组：

$$\text{OldChrom}=\begin{bmatrix}40.23 & -17.17 & 28.95 & 15.38\\ 82.06 & 13.26 & 13.35 & -9.09\\ 52.43 & 25.64 & 15.20 & -2.54\end{bmatrix}\begin{matrix}\%\text{父代 1}\\ \%\text{父代 2}\\ \%\text{父代 3}\end{matrix}$$

执行线性重组：

NewChrom＝reclin (OldChrom)

通过向最先的父个体增加不同微量产生新值，中间比率因子表 Alpha 被产生，例如：

$$\text{Alpha}=\begin{bmatrix}0.78\\ 1.05\end{bmatrix}\begin{matrix}\%\text{子代 1}\\ \%\text{子代 2}\end{matrix}$$

因此，重组后的 NewChrom 成为

$$\text{NewChrom}=\begin{bmatrix}72.97 & 6.64 & 16.74 & -3.77 \\ 84.25 & 14.85 & 12.54 & -10.37\end{bmatrix} \quad \begin{matrix}\%\ \text{Alpha(1，:)，父 1 和 2} \\ \%\ \text{Alpha(2，:)，父 1 和 2}\end{matrix}$$

由于父种群 OldChrom 的个体数是奇数，最后一个个体不参加重组直接加入 NewChrom，因此返回用户空间的后代如下：

$$\text{NewChrom}=\begin{bmatrix}72.97 & 6.64 & 16.74 & -3.77 \\ 84.25 & 14.85 & 12.54 & -10.37 \\ 52.43 & 25.64 & 15.20 & -2.54\end{bmatrix}$$

算法说明：线性重组重组父值使用如下公式：

offspring＝parent1＋Alpha ＊（parent2－parent1）

这里，Alpha 是在区间[－0.25，1.25]内随机一致性选择产生的标量因子。reclin 对重组的每一对双亲产生一新的 Alpha。

线性重组能产生略大于双亲定义的线段中的任意点。

线性重组 reclin 与中间重组 recint 相似。然而 reclin 对每对双亲使用一 Alpha 因子一起重组，而 recint 对每对值使用了一新的 Alpha 值。

参见：recombin，recdis，recint，ranking，sus，rws。

### 5.2.13 函数 recmut

功能：具有突变特征的线性重组。

格式：① NewChrom ＝ recmut(OldChrom，FieldDR)

② NewChrom ＝ recmut(OldChrom，FieldDR，MutOpt)

详细说明：recmut 完成当前种群 OldChrom 成对个体的具有突变特征的线性重组，返回交配后的新种群 NewChrom。OldChrom 中每一行对应一个个体。

FieldDR 是一矩阵，包含一个个体的每个变量的边界(参见 crtrp)。

MutOpt 是一最多有两个参数的任选向量。

MutOpt(1)是一包含在[0，1]范围内的重组概率的标量。如果省略或为 NaN，则 MutOpt(1)假设为 1。

MutOpt(2)是一包含在[0，1]范围内用于压缩重组范围的标量值。如果省略或为 NaN，则 MutOpt(2)假设为 1(不压缩)。

recmut 是一个只能应用于实值(非二进制、非整数)变量种群的函数。

交配的对是有序的，奇数行与它下一个偶数行配对。如果 OldChrom 的行数是奇数，则最后一个奇数行不参与交配，直接加到 NewChrom 的末尾。因此种群根据交配要求组织成连续的对。通过使用 ranking 计算每个个体的适应度水平，由选择函数(例如select)用与适应度相关的概率在当前种群中选择个体，完成这个重组工作。

recmut 是一使用突变特征因子的遗传算法育种器(参见 mutbga)。这个重组函数的调用语法与突变函数 mutbga 的语法相同。

recmut 是一低级重组函数，通常被 mutate 调用。

例 **5.13** 函数 recmut 的应用举例。下面为具有 4 个实值个体的种群的线性重组示例。

$$\text{OldChrom}=\begin{bmatrix} 40.2381 & -17.1766 & 28.9530 & 15.3883 \\ 82.0642 & 13.2639 & 13.3596 & -9.0916 \\ 52.4396 & 25.6410 & 15.2014 & -2.5435 \\ -47.5381 & 49.1963 & 9.0954 & 10.6521 \end{bmatrix} \begin{matrix} \% \text{ p1} \\ \% \text{ p2} \\ \% \text{ p3} \\ \% \text{ p4} \end{matrix}$$

边界定义如下：

$$\text{FieldDR}=\begin{bmatrix} -100 & -50 & -30 & -20 \\ 100 & 50 & 30 & 20 \end{bmatrix}$$

执行下列具有突变特征的线性重组：

NewChrom＝recmut(OldChrom，FieldDR)

recmut 产生一中间任务表 RecMx，决定哪些对双亲参加重组(这里全部参加)并记下重组的步长。例如：

$$\text{RecMx}=\begin{bmatrix} 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 \end{bmatrix} \begin{matrix} \%\text{p1\&p2} \\ \%\text{p3\&p4} \end{matrix}$$

两个更进一步的表 delta 和 Diff 指出正常的重组步长：

$$\text{delta}=\begin{bmatrix} 0.1250 & 0.1250 & 0.1250 & 0.1250 \\ 0.0005 & 0.0005 & 0.0005 & 0.0005 \end{bmatrix} \begin{matrix} \%\text{p1\&p2} \\ \%\text{p3\&p4} \end{matrix}$$

$$\text{Diff}=\begin{bmatrix} 1.3937 & 1.0143 & -0.5196 & -0.8157 \\ -10.5712 & 2.4906 & -0.6456 & 1.3952 \end{bmatrix} \begin{matrix} \%\text{p1\&p2} \\ \%\text{p3\&p4} \end{matrix}$$

重组后 NewChrom 成为

$$\text{NewChrom}=\begin{bmatrix} 57.6637 & -23.5177 & 30.0000 & 17.4281 \\ 64.6386 & 19.6050 & 11.4106 & -11.1314 \\ 52.9719 & 25.5783 & 15.2112 & -2.5576 \\ -48.0704 & 49.2590 & 9.0856 & 10.6662 \end{bmatrix}$$

算法说明：一对双亲的后代按如下方法计算：

offspring1＝parent1 ＋ RecMx×range×MutOpt(2)×delta×Diff

offspring2＝parent2 ＋ RecMx×range×MutOpt(2)×delta×(－Diff)

其中

$$\text{delta} = \sum_{i=0}^{m-1} a_i 2^{-i}$$

RecMx ＝±1，具有概率 MutOpt(1)(在这里为 0.9)，否则为 0。

range＝0.5，为变量的域（参看 FieldDR 的定义)。

$a_i=1$，具有概率 $1/m$，否则为 0，其中 $m=20$。

$$\text{Diff}=\frac{\text{parent1}-\text{parent2}}{\|\text{parent1}-\text{parent2}\|}$$

这个重组算子产生其双亲(线性重组)定义方向的子代。它经常超出双亲定义的范围或一个父亲定义的方向。子代的这点是由突变算子的特征定义的，小步长情况产生的概率要比大步长产生的概率大(参看 mutbga)。

参见：mutate，mutbga，reclin。

### 5.2.14 函数 recombin

功能：重组个体(高级函数)。

格式：① NewChrom＝recombin(REC_F，Chrom)

② NewChrom ＝ recombin(REC_F，Chrom，RecOpt)

③ NewChrom ＝ recombin(REC_F，Chrom，RecOpt，SUBPOP)

详细说明：recombin 完成种群 Chrom 中个体的重组，在新种群 NewChrom 中返回重组后的个体。Chrom 和 NewChrom 中的一行对应一个个体。

REC_F 是一包含低级重组函数名的字符串，例如 recdis 或 xovsp。

RecOpt 是一指明交叉概率的任选参数，如省略 RecOpt 或为 NaN，将设为缺省值。

SUBPOP 是一决定 Chrom 中子种群个数的可选参数，如果省略 SUBPOP 或为 NaN，则假设 SUBPOP＝1。Chrom 中的所有子种群必须有相同的大小。

例 **5.14** 函数 recombin 的应用举例。下面的程序代码为：首先产生 5 个个体的种群，然后利用函数 recombin 对该种群进行重组。

$$\text{BaseV}=\begin{bmatrix} -100 & -50 & -30 & -20 \\ 100 & 50 & 30 & 20 \end{bmatrix}$$

Chrom＝crtrp(5，BaseV)；% 产生 5 个个体的种群

$$\text{Chrom}=\begin{bmatrix} 13.3546 & -44.1138 & -27.0433 & 9.5997 \\ 64.6017 & -13.9689 & 4.2634 & -2.7251 \\ 34.7897 & 4.8513 & 12.0514 & 5.3706 \\ 99.8895 & -23.8230 & 27.7373 & 12.1211 \\ 92.3273 & 9.7345 & 15.0311 & -16.6448 \end{bmatrix}$$

NewChrom＝recombin('recdis'，Chrom)；　% 参数选取离散重组

$$\text{NewChrom}=\begin{bmatrix} 64.6017 & -13.9689 & 4.2634 & -2.7251 \\ 64.6017 & -13.9689 & 4.2634 & -2.7251 \\ 99.8895 & 4.8513 & 27.7373 & 5.3706 \\ 34.7897 & 4.8513 & 12.0514 & 5.3706 \\ 92.3273 & 9.7345 & 15.0311 & -16.6448 \end{bmatrix}$$

NewChrom＝recombin('xovsp'，Chrom)；　% 参数选取单点交叉

$$\text{NewChrom}=\begin{bmatrix} 13.3546 & -44.1138 & 4.2634 & -2.7251 \\ 64.6017 & -13.9689 & -27.0433 & 9.5997 \\ 34.7897 & 4.8513 & 27.7373 & 12.1211 \\ 99.8895 & -23.8230 & 12.0514 & 5.3706 \\ 92.3273 & 9.7345 & 15.0311 & -16.6448 \end{bmatrix}$$

算法说明：recombin 检测输入参数的一致性并调用低级重组函数。如果 recombin 调用时具有多个子种群，则对每个子种群分别调用低级重组函数。

参见：recdis，recint，reclin，xovsp，xovdp，xovsh，mutate，select。

### 5.2.15 函数 reins

功能：重插入子代到种群。

格式：① Chrom＝reins(Chrom，SelCh)

② Chrom＝reins(Chrom，SelCh，SUBPOP)

③ Chrom＝reins(Chrom，SelCh，SUBPOP，InsOpt，ObjVCh)

④ [Chrom，ObjVCh]＝reins(Chrom，SelCh，SUBPOP，InsOpt，ObjVCh，ObjVSel)

详细说明：reins 完成插入子代到当前种群，用子代代替父代并返回结果种群。子代包含在矩阵 SelCh 中，父代在矩阵 Chrom 中，Chrom 和 Selch 中每一行对应一个个体。

SUBPOP 是一可选参数，指明 Chrom 和 SelCh 中子种群的个数。如果 SUBPOP 省略或为 NaN，则假设 SUBPOP＝1。在 Chrom 和 SelCh 中每个子种群必须具有相同大小。

InsOpt 是一最多有两个参数的任选向量。

InsOpt(1)是一标量，指明用子代代替父代的选择方法。0 为均匀选择，子代代替父代使用均匀随机选择。1 为基于适应度的选择，子代代替最小适应的个体。如果省略 InsOpt(1)或为 NaN，则假设 InsOpt(1)＝0。

InsOpt(2)是一在[0，1]间的标量，表示每个子种群中重插入的子代个体在整个子种群中个体的比率。如果省略 InsOpt(2)或为 NaN，则假设 InsOpt(2)＝1.0。

如果 InsOpt 省略或为 NaN，则 InsOpt 为缺省值。

ObjVCh 是一可选的列向量，包含 Chrom 中个体的目标值。对基于适应度的重插入，ObjVCh 是必需的。

ObjVSel 是一可选的列向量，包含 Selch 中个体的目标值。如果子代的数量大于重插入种群中的子代数量，则 ObjVSel 是必需的。在这种情况下，子代将按它们的适应度选择插入。

如果 ObjVCh 是一输出参数，ObjVCh 和 ObjVSel 需要作为输入参数，随后目标被拷贝，按照重插入的子代，保存为整修种群重新计算的目标值。

例 **5.15** 函数 reins 的应用示例。下面的程序代码为在 6 个个体的父代种群中插入子代种群。

```
FieldDR1=[-10，-5，-3，-1；10，5，3，1]；    % 定义边界变量
```

$$\mathrm{FieldDR1}=\begin{bmatrix}-10 & -5 & -3 & -1\\ 10 & 5 & 3 & 1\end{bmatrix}$$

```
Chrom=crtrp(6，FieldDR1)；    % 产生 6 个个体的父代种群
```

$$\mathrm{Chrom}=\begin{bmatrix}-5.7602 & 1.6612 & 2.9129 & 0.6440\\ -0.0318 & -3.6906 & -2.8958 & -0.4736\\ -4.1902 & -4.0459 & 1.9164 & 0.5073\\ 3.4551 & -4.8514 & 0.7268 & 0.3193\\ 9.1598 & -2.1181 & 0.3613 & -0.5719\\ 5.3310 & 3.1673 & -1.5358 & 0.2042\end{bmatrix}$$

```
FieldDR2=[-100，-50，-30，-20；100，50，30，20]；    % 定义边界变量
```

$$\text{FieldDR2}=\begin{bmatrix} -100 & -50 & -30 & -20 \\ 100 & 50 & 30 & 20 \end{bmatrix}$$

SelCh＝crtrp(2，FieldDR2)；　　% 产生 2 个个体的子代种群

$$\text{SelCh}=\begin{bmatrix} -42.7701 & 43.2736 & 26.4492 & 13.9071 \\ -49.7599 & -36.9018 & 12.1112 & -11.6291 \end{bmatrix}$$

插入所有子代到种群中：

Chrom ＝ reins(Chrom，SelCh)

这个新种群 Chrom 产生：

$$\text{Chrom}=\begin{bmatrix} -5.7602 & 1.6612 & 2.9129 & 0.6440 \\ -0.0318 & -3.6906 & -2.8958 & -0.4736 \\ -49.7599 & -36.9018 & 12.1112 & -11.6291 \\ 3.4551 & -4.8514 & 0.7268 & 0.3193 \\ -42.7701 & 43.2736 & 26.4492 & 13.9071 \\ 5.3310 & 3.1673 & -1.5358 & 0.2042 \end{bmatrix}$$

为父种群 Chrom 考虑如下目标值 ObjVCh 向量，为子代 Selch 考虑如下目标值 ObjVSel 向量：

ObjVCh＝[21；22；23；24；25；26]；

ObjVSel＝[31；32]；

基于适应度插入所有子代代替最不适应的父个体。

Chrom＝reins(Chrom，SelCh，1，1，ObjVCh)；

$$\text{Chrom}=\begin{bmatrix} -5.7602 & 1.6612 & 2.9129 & 0.6440 \\ -0.0318 & -3.6906 & -2.8958 & -0.4736 \\ -4.1902 & -4.0459 & 1.9164 & 0.5073 \\ 3.4551 & -4.8514 & 0.7268 & 0.3193 \\ -49.7599 & -36.9018 & 12.1112 & -11.6291 \\ -42.7701 & 43.2736 & 26.4492 & 13.9071 \end{bmatrix}$$

基于适应度插入 50％的子代，并按插入的子代拷贝目标值：

[Chrom，ObjVCh]＝reins(Chrom，SelCh，1，[1 0.5]，ObjVCh，ObjVSel)

ObjVCh＝[21；22；23；24；25；31]

$$\text{Chrom}=\begin{bmatrix} -5.7602 & 1.6612 & 2.9129 & 0.6440 \\ -0.0318 & -3.6906 & -2.8958 & -0.4736 \\ -4.1902 & -4.0459 & 1.9164 & 0.5073 \\ 3.4551 & -4.8514 & 0.7268 & 0.3193 \\ 9.1598 & -2.1181 & 0.3613 & -0.5719 \\ -42.7701 & 43.2736 & 26.4492 & 13.9071 \end{bmatrix}$$

利用函数 reins，将两个子种群 Chrom 和 Selch 插入到当前种群。

Chrom＝reins(Chrom，SelCh，2)

$$\text{Chrom}=\begin{bmatrix} -5.7602 & 1.6612 & 2.9129 & 0.6440 \\ -42.7701 & 43.2736 & 26.4492 & 13.9071 \\ -4.1902 & -4.0459 & 1.9164 & 0.5073 \\ 3.4551 & -4.8514 & 0.7268 & 0.3193 \\ 9.1598 & -2.1181 & 0.3613 & -0.5719 \\ -49.7599 & -36.9018 & 12.1112 & -11.6291 \end{bmatrix}$$

参见：select。

### 5.2.16 函数 rep

功能：矩阵复制。

格式：MatOut=rep(MatIn，REPN)

详细说明：rep 是一个低级复制函数，通常不直接使用，它可被 GA 工具箱中许多函数调用。

rep 完成矩阵 MatIn 的复制，指明复制次数 REPN，返回复制后的矩阵 MatOut。REPN 包含每个方向的复制次数。REPN(1)指明纵向复制次数，REPN(2)指明水平方向复制次数。

例 **5.16** 矩阵复制函数 rep 的应用示例。考虑以下矩阵 MatIn：

$$\text{MatIn}=\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{bmatrix}$$

执行下列复制：

(1) MatOut = rep(MatIn，[1 2])；

$$\text{MatOut}=\begin{bmatrix} 1 & 2 & 3 & 4 & 1 & 2 & 3 & 4 & & & & \\ 5 & 6 & 7 & 8 & 5 & 6 & 7 & 8 & 5 & 6 & 7 & 8 \end{bmatrix}$$

(2) MatOut = rep(MatIn，[2 1])；

$$\text{MatOut}=\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{bmatrix}$$

(3) MatOut = rep(MatIn，[2 3])；

$$\text{MatOut}=\begin{bmatrix} 1 & 2 & 3 & 4 & 1 & 2 & 3 & 4 & 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 & 5 & 6 & 7 & 8 & 5 & 6 & 7 & 8 \\ 1 & 2 & 3 & 4 & 1 & 2 & 3 & 4 & 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 & 5 & 6 & 7 & 8 & 5 & 6 & 7 & 8 \end{bmatrix}$$

### 5.2.17 函数 rws

功能：轮盘赌选择。

格式：NewChrIx=rws(FitnV，Nsel)

详细说明：rws 在当前种群中按照它们的适应度 FitnV 选择 Nsel 个个体繁殖。

NewChrIx = rws(FitnV，Nsel)使用轮盘赌选择从一个种群中选择 Nsel 个个体。

FitnV 是一包含种群中每个个体性能尺寸的列向量，它能通过使用函数 ranking 或 scaling 计算每个个体的适应度水平来得到。返回值 NewChrIx 是为育种选择的个体的索引值，按照它们的选择顺序排列，这些选择的个体能通过评估函数 Chrom(NewChrIx，:)恢复。

rws 是一低级选择函数，通常被 select 调用。

例 **5.17**　轮盘赌选择方法示例。考虑 8 个个体的种群，假设已计算出适应度 FitnV：

FitnV=[1.50；1.35；1.21；1.07；0.92；0.78；0.64；0.5]

选择 6 个个体的索引：

NewChrIx=rws(FitnV，6)

NewChrIx 成为

$$\text{NewChrIx}=\begin{bmatrix}2\\5\\1\\1\\3\\7\end{bmatrix}$$

算法说明：通过计算适应度向量的累加和完成轮盘赌选择的表格，并产生随机分布在[0，sum(FitnV)]区间内的 Nsel 个实数，被选择个体的索引通过比较向量累加和产生的编号来决定。一个个体被选择的概率由下式给出：

$$F(x_i)=\frac{f(x_i)}{\sum_{i=1}^{N_{ind}} f(x_i)}$$

这里，$f(x_i)$是个体 $x_i$ 的适应度，$F(x_i)$是这个个体被选择的概率。

参见：select，sus，reins，ranking，scaling。

### 5.2.18　函数 scaling

功能：线性适应度计算。

格式：FitnV=scaling(ObjV，Smul)

详细说明：scaling 转换一种群的目标值 ObjV 为由 Smul 的值决定上界的适应度值。例如：

$$F(x_i)=af(x_i)+b$$

这里 $f(x_i)$是个体 $x_i$ 的适应度，$a$ 是一标量系数，$b$ 是一偏移值，$F(x_i)$是这个个体产生的适应度值。

如果 $f_{ave}$ 是当前代中目标值的平均值，种群中最大适应度值是上界 $f_{ave}\cdot$Smul，如果省略 smul，则 smul 假设为 2，这个种群的平均适应度值也为 $f_{ave}$。

例 **5.18**　目标函数值的线性适应度计算示例。

(1) ObjV=[1；2；3；4；5；6]

FitnV=scaling(ObjV)

$$\text{FitnV}=\begin{bmatrix}0\\1.4000\\2.8000\\4.2000\\5.6000\\7.0000\end{bmatrix}$$

在一些情况下，一些目标值是负的，scaling 试图提供偏移值 $b$，确保计算适应度值大于 0。

（2）ObjV＝[1；2；4；3；9；13；5；6]

FitnV＝scaling(ObjV)

delta＝7.6250;a＝0.7049;b＝1.5861。

$$\text{FitnV}=\begin{bmatrix}2.2910\\2.9959\\4.4057\\3.7008\\7.9303\\10.7500\\5.1107\\5.8156\end{bmatrix}$$

算法说明：scaling 使用 Goldberg 描述的线性比率法。

注意：线性比率不适合目标函数返回负的适应度值的情形。

参见：ranking，reins，rws，select，sus。

### 5.2.19 函数 select

功能：从种群中选择个体(高级函数)。

格式：① SelCh＝select(SEL_F，Chrom，FitnV)

② SelCh＝select(SEL_F，Chrom，FitnV，GGAP)

③ SelCh＝select(SEL_F，Chrom，FitnV，GGAP，SUBPOP)

详细说明：利用函数 select 从种群 Chrom 中选择优良个体，并将选择的个体返回到新种群 SelCh 中，Chrom 和 SelCh 中每一行对应一个个体。

SEL_F 是一字符串，包含一低级选择函数名，如 rws 或 sus。

FitnV 是一列向量，包含种群 Chrom 中个体的适应度值。这个适应度值表明了每个个体被选择的预期概率。

GGAP 是一可选参数，指出了代沟，部分种群被复制。如果 GGAP 缺省或为 NaN，则 GGAP 假设为 1.0。GGAP 也可大于 1，允许子代数多于父代的数量，如 Chrom 超过一个子种群，GGAP 指明每个子种群被选择的个体数量是与子种群的大小有关的。

SUBPOP 是一可选参数，决定 Chrom 中子种群的数量。如果 SUBPOP 省略或为 NaN，则 SUBPOP＝1。Chrom 中所有子种群必须有相同的大小。

例 **5.19** 函数 select 的应用举例。考虑以下具有 8 个个体的种群 Chrom，适应度值为

FitnV：

$$Chrom=\begin{bmatrix}1 & 11 & 21\\2 & 12 & 22\\3 & 13 & 23\\4 & 14 & 24\\5 & 15 & 25\\6 & 16 & 26\\7 & 17 & 27\\8 & 18 & 28\end{bmatrix}$$

FitnV=[1.50；1.35；1.21；1.07；0.92；0.78；0.64；0.5]

使用随机遍历抽样 sus 选择 8 个个体：

SelCh=select('sus'，Chrom，FitnV)

SelCh 成为

$$Selch=\begin{bmatrix}7 & 17 & 27\\1 & 11 & 21\\6 & 16 & 26\\1 & 11 & 21\\5 & 15 & 25\\2 & 12 & 22\\3 & 13 & 23\\4 & 14 & 24\end{bmatrix}$$

假设 Chrom 由两个子种群组成，通过轮盘赌选择 sus 对每个子种群选择 150% 的个体。

FitnV=[1.50；1.16；0.83；0.50；1.50；1.16；0.83；0.5]

SelCh=select('sus'，Chrom，FitnV，1.5，2)

SelCh 变成

$$Selch=\begin{bmatrix}3 & 13 & 23\\2 & 12 & 22\\1 & 11 & 21\\2 & 12 & 22\\2 & 12 & 22\\1 & 11 & 21\\6 & 16 & 26\\7 & 17 & 27\\7 & 17 & 27\\6 & 16 & 26\\7 & 17 & 27\\5 & 15 & 25\end{bmatrix}$$

算法说明：select 检测输入参数的一致性并调用低级选择函数，如果调用 select 使用

多子种群，则低级选择函数分别被各子种群调用。

参见：rws，sus，ranking，scaling，recombin，mutate。

### 5.2.20 函数 sus

功能：随机遍历抽样。

格式：NewChrIx=sus(FitnV，Nsel)

详细说明：sus 按照个体在当前种群中的适应度 FitnV 为繁殖概率性选择 Nsel 个个体。

NewChrIx = sus(FitnV，Nsel)使用随机遍历抽样从种群选择 Nsel 个个体。FitnV 是一列向量，包含种群中个体的适应度值，它可通过函数 ranking 或 scaling 计算种群中个体适应度水平得到。返回值 NewChrIx 是为培养选择的个体索引值，是按它们选择的顺序排列的。这个选择的个体可通过评估 Chrom(NewChrIx，:)恢复。

sus 是一低级选择函数，通常被 select 调用。

例 **5.20** 随机遍历抽样函数 sus 的应用举例。考虑以下具有 8 个个体的种群 Chrom，适应度值为 FitnV：

FitnV=[1.50；1.35；1.21；1.07；0.92；0.78；0.64；0.5]

选择 6 个个体的索引：

NewChrIx=sus(FitnV，6)

NewChrIx 成为

$$\text{NewChrIx}=\begin{bmatrix}2\\5\\3\\1\\7\\4\end{bmatrix}$$

算法说明：通过获得适应度向量 FitnV 的累加和完成随机遍历抽样的表格，产生 Nsel 个在[0，SUM(FitnV)]内的相等空间编号。因为只有一个随机数产生，所以其他使用是来自那些点的相等空间。被选择个体的索引是通过比较产生的数与向量累加和来决定的。一个个体被选择的概率由下式给出：

$$F(x_i)=\frac{f(x_i)}{\sum_{i=1}^{N_{ind}} f(x_i)}$$

这里，$f(x_i)$是个体 $x_i$ 的适应度，$F(x_i)$是这个个体被选择的概率。

参见：select，rws，reins，ranking，scaling。

### 5.2.21 函数 xovdp

功能：两点交叉。

格式：NewChrom = xovdp(OldChrom，XOVR)

详细说明：xovdp 完成当前种群 OldChrom 中一对个体按交叉概率 XOVR 进行交叉，

返回交配后的新种群 NewChrom。OldChrom 和 NewChrom 的一行对应一个个体，它可用于任何染色体表示。

XOVR 是一可选参数，说明交叉概率，如果省略、空或为 NaN，则假设 XOVR＝0.7。

交配的对是有序的，即奇行与它下一个偶行配对。如果矩阵 OldChrom 行数是奇数行，则最后一行不参加交配，因此，种群将按交配要求组织成连续的对。这可使用函数 ranking 计算每个染色的适应度，并由选择函数(select，sus 或 rws)在种群中用与适应度相关的概率选择个体来完成。

xovdp 是一低级交叉函数，通常被函数 recombin 调用。

例 **5.21**　两点交叉函数 xovdp 的应用举例。下面的程序代码为：首先创建初始种群，然后利用函数 xovdp 进行两点交叉。

Chrom＝crtbp(5，6)；　% 创建初始种群

$$\text{Chrom}=\begin{bmatrix}1&0&1&1&1&0\\0&1&0&1&1&1\\0&0&1&1&0&1\\0&0&1&1&1&1\\1&1&0&0&1&0\end{bmatrix}$$

NewChrom＝xovdp(Chrom，0.7)；

运行后得

$$\text{NewChrom}=\begin{bmatrix}0&0&0&0&1&1\\0&1&0&0&0&0\\1&0&1&0&0&1\\0&0&1&1&1&1\\0&1&0&0&1&1\end{bmatrix}$$

算法说明：考虑下面两个相同长度的二进制串：

A1＝[1 1 0 1 0 1]

A2＝[1 0 1 0 1 0]

两点交叉包括选择两个均匀分布的随机整数 k1，k2，在[1，leng(A1)]范围内、在 A1 和 A2 间交换 k1＋1 到 k2 之间的各变量，如果这个交叉 k1＝3，k2＝5，则 A1，A2 成为

$A1^{*}$ ＝ [1 1 0 0 1 1]

$A2^{*}$ ＝ [1 0 1 1 0 0]

xovdp 用适当的参数调用 xovmp。

参见：xovdprs，xovsp，xovsh，xovmp，recombin，select。

### 5.2.22　函数 xovdprs

功能：减少代理的两点交叉。

格式：NewChrom ＝ xovdprs(OldChrom，XOVR)

详细说明：xovdprs 在当前种群 OldChrom 一对个体间按交叉概率 XOVR 进行减少代理两点交叉，并返回交配后的新种群 NewChrom。在 OldChrom 和 NewChrom 中每一行对应一个个体。它适用于任意染色体表示。

XOVR 是一可选参数，指明交叉概率，如果省略、空或为 NaN，则设 XOVR＝0.7。

交配的对是有序的，即奇行与它下一个偶行配对。如果矩阵 OldChrom 行数是奇数行，则最后一行不参加交配，因此，种群将按交配要求组织成连续的对。这可使用函数 ranking 计算每个染色体的适应度，并由选择函数（select，sus 或 rws）在种群中用与适应度相关的概率选择个体来完成。

xovdprs 是一低级交叉函数，通常被函数 recombin 调用。

例 **5.22** 减少代理的两点交叉函数 xovdprs 的应用举例。下面的程序代码为 5 个个体种群的两点交叉示例：

```
Chrom=crtbp(5, 6);      %创建初始种群
```

$$Chrom=\begin{bmatrix}1&0&1&1&1&0\\0&1&0&1&1&1\\0&0&1&1&0&1\\0&0&1&1&1&1\\1&1&0&0&1&0\end{bmatrix}\qquad \%\ 5\ 个个体的种群$$

```
NewChrom=xovdprs(Chrom, 0.7);
```

运行后得

$$NewChrom=\begin{bmatrix}0&0&0&0&1&0\\0&1&0&0&0&1\\1&0&1&0&0&1\\0&0&1&1&1&1\\0&1&0&0&1&1\end{bmatrix}$$

算法说明：参见两点交叉函数 xovdp。

交叉对任何可能点总是产生新个体，减少代理算子限制它。它通过限制交叉点的位置来完成，例如，交叉点只能发生在基因不同的地方。

xovdprs 使用适当参数调用 xovmp。

参见：xovdp，xovsprs，xovshrs，xovmp，recombin，select。

### 5.2.23 函数 xovmp

功能：多点交叉。

格式：NewChrom = xovmp(OldChrom，XOVR，Npt，Rs)

详细说明：xovmp 在当前种群 OldChrom 成对个体间多点交叉，并返回交配后的新种群 NewChrom。在 OldChrom 和 NewChrom 中每一行对应一个体。它适用于任意染色体表示。

XOVR 是一可选参数，指明交叉概率，如果省略、为空或 NaN，则设 XOVR＝0.7。

Npt 是一可选参数，指明交叉点数。0—洗牌交叉；1—单点交叉；2—两点交叉。如果省略或为 NaN，则假设 Npt＝0。

Rs 是一可选参数，指明使用减少代理。0—不减少代理；1—减少代理。如果省略、空或为 NaN，则设 Rs＝0。

交配的对是有序的，即奇行与它下一个偶行配对。如果矩阵 OldChrom 行数是奇数行，

则最后一行不参加交配，因此，种群将按交配要求组织成连续的对。这可使用函数 ranking 计算每个染色体的适应度，并由选择函数(select，sus 或 rws)在种群中用与适应度相关的概率选择个体来完成。

xovmp 是一低级交叉函数，通常被其他交叉函数调用。如果被 recombine、xovmp 调用执行没有减少代理的洗牌交叉，则等同于 xovsh。

例 **5.23** 多点交叉函数 xovmp 的应用示例。下面的程序代码为 5 个个体种群的多点交叉示例：

```
Chrom=crtbp(5，6)；        % 创建初始种群
```

$$\text{Chrom}=\begin{bmatrix}1&0&1&1&1&0\\0&1&0&1&1&1\\0&0&1&1&0&1\\0&0&1&1&1&1\\1&1&0&0&1&0\end{bmatrix}\qquad \%\ 5\ \text{个个体的种群}$$

```
NewChrom=xovmp(Chrom，0.7，1，1)；
```

运行后得

$$\text{NewChrom}=\begin{bmatrix}0&0&0&0&0&0\\0&1&0&0&1&1\\1&0&1&1&1&1\\0&0&1&0&0&1\\0&1&0&0&1&1\end{bmatrix}$$

算法说明：多点交叉函数的使用可参考单点交叉 xovsp、两点交叉 xovdp 和洗牌交叉 xovsh，也可参考具有减少代理的单点交叉 xovsprs、两点交叉 xovdprs 和洗牌交叉 xovshrs。

参见：xovsp，xovdp，xovsh，xovsprs，xovdprs，xovshrs，recombin。

### 5.2.24 函数 xovsh

功能：洗牌交叉。

格式：NewChrom = xovsh(OldChrom，XOVR)

详细说明：xovsh 在当前种群 OldChrom 一对个体间按交叉概率 XOVR 进行洗牌交叉，并返回交配后的新种群 NewChrom。在 OldChrom 和 NewChrom 中每一行对应一个个体。它适用于任意染色体表示。

XOVR 是一可选参数，指明交叉概率，如果省略、为空或 NaN，则设 XOVR=0.7。

交配的对是有序的，即奇行与它下一个偶行配对。如果矩阵 OldChrom 行数是奇数行，则最后一行不参加交配，因此，种群将按交配要求组织成连续的对。这可使用函数 ranking 计算每个染色体的适应度，并由选择函数(select，sus 或 rws)在种群中用与适应度相关的概率选择个体来完成。

xovsh 是一低级交叉函数，通常被函数 recombin 调用。

例 **5.24** 洗牌交叉函数 xovsh 的应用举例。下面的程序代码为 5 个个体种群的洗牌交叉示例：

Chrom＝crtbp(5，6)；　　　% 创建初始种群

$$Chrom=\begin{bmatrix}0&1&1&0&1&0\\1&1&1&0&1&1\\0&1&1&1&0&1\\1&0&1&1&1&1\\0&1&0&0&1&1\end{bmatrix}$$　　% 5 个个体的种群

NewChrom＝xovsh(Chrom，0.7)；

运行后得

$$NewChrom=\begin{bmatrix}0&0&1&1&0&0\\1&1&1&0&0&0\\1&0&1&1&1&1\\0&1&1&0&0&0\\1&0&0&1&0&0\end{bmatrix}$$

算法说明：洗牌交叉是一种单点交叉，但在位交换之间，它们可在双亲间随意洗牌。在重组后，子代的各位是没有洗牌的，这种移位就像在每次交叉时，位可任意再分配。

xovsh 使用适当的参数调用 xovmp。

参见：xovshrs，xovsp，xovdp，xovmp，recombin，select。

### 5.2.25　函数 xovshrs

功能：减少代理的洗牌交叉。

格式：NewChrom ＝ xovshrs(OldChrom，XOVR)

详细说明：xovshrs 在当前种群 OldChrom 一对个体间按交叉概率 XOVR 进行减少代理的洗牌交叉，并返回交配后的新种群 NewChrom。在 OldChrom 和 NewChrom 中每一行对应一个体。它适用于任意染色体表示。

XOVR 是一可选参数，指明交叉概率，如果省略、为空或 NaN，则设 XOVR＝0.7。

交配的对是有序的，即奇行与它下一个偶行配对。如果矩阵 OldChrom 行数是奇数行，则最后一行不参加交配，因此，种群将按交配要求组织成连续的对。这可使用函数 ranking 计算每个染色体的适应度，并由选择函数(select，sus 或 rws)在种群中用与适应度相关的概率选择个体来完成。

xovshrs 是一低级交叉函数，通常被函数 recombin 调用。

例 **5.25**　减少代理的洗牌交叉函数 xovshrs 的应用举例。下面的程序代码为 5 个个体种群的减少代理的洗牌交叉示例：

NewChrom＝crtbp(5，6)；　　　% 创建初始种群

$$Chrom=\begin{bmatrix}0&0&1&1&1&0\\1&0&1&0&0&1\\1&0&0&1&0&1\\0&1&0&1&0&1\\1&1&1&0&1&1\end{bmatrix}$$　　% 5 个个体的种群

NewChrom＝xovshrs(Chrom，0.7)；

运行后得

$$NewChrom=\begin{bmatrix}0&0&1&1&0&0\\1&1&1&0&0&0\\1&1&1&1&0&1\\0&0&1&0&1&0\\1&0&0&1&0&0\end{bmatrix}$$

算法说明：洗牌交叉算法参见 xovsh。

交叉对任何可能点总是产生新个体，减少代理算子限制它。它是通过限制交叉点的位置来完成的，例如，交叉点只能发生在基因不同的地方。

xovshrs 使用适当参数调用 xovmp。

参见：xovsh，xovsprs，xovdprs，xovmp，recombin，select。

### 5.2.26 函数 xovsp

功能：单点交叉。

格式：NewChrom＝xovsp(OldChrom，XOVR)

详细说明：xovsp 完成当前种群 OldChrom 中一对个体按交叉概率 XOVR 进行单点交叉，返回交配后的新种群 NewChrom。OldChrom 和 NewChrom 的一行对应一个个体，它可用于任何染色体表示。

XOVR 是一可选参数，说明交叉概率，如果省略、空或为 NaN，则假设 XOVR＝0.7。

交配的对是有序的，即奇行与它下一个偶行配对。如果矩阵 OldChrom 行数是奇数行，则最后一行不参加交配，因此，种群将按交配要求组织成连续的对。这可使用函数 ranking 计算每个染色体的适应度，并由选择函数(select，sus 或 rws)在种群中用与适应度相关的概率选择个体来完成。

xovsp 是一低级交叉函数，通常被函数 recombin 调用。

例 **5.26**　单点交叉函数 xovsp 的应用举例。下面的程序代码为：首先创建 2 个个体的种群，然后利用函数 xovsp 进行单点交叉：

```
Chrom=crtbp(2,8);        % 创建2个个体的种群
```

$$Chrom=\begin{bmatrix}1&1&0&0&0&0&1&1\\0&1&0&1&1&1&0&1\end{bmatrix}$$

```
NewChrom=xovsp(Chrom,0.5);      % 交叉概率为0.5
```

则 Chrom 成为

$$NewChrom=\begin{bmatrix}1&1&0&0&0&1&0&1\\0&1&0&1&1&0&1&1\end{bmatrix}$$

```
NewChrom=xovsp(Chrom,3);     % 交叉点 k=3
```

则 Chrom 成为

$$NewChrom=\begin{bmatrix}1&1&0&1&1&1&0&1\\0&1&0&0&0&0&1&1\end{bmatrix}$$

算法说明：考虑下面两个相同长度的二进制串：

A1＝[1 1 0 1 0 1]

A2=[1 0 1 0 1 0]

单点交叉包括选择一个均匀分布的随机整数 k，在[1，leng(A1)-1]间、在 A1 和 A2 间交换 k+1 到 leng(A1)之间的各变量，如果这个交叉 k=3，则 A1，A2 成为

A1*=[1 1 0 0 1 0]

A2*=[1 0 1 1 0 1]

xovsp 使用适当的参数调用 xovmp。

参见：xovsprs，xovdp，xovsh，xovmp，recombin，select。

### 5.2.27 函数 xovsprs

功能：减少代理的单点交叉。

格式：NewChrom = xovsprs(OldChrom，XOVR)

详细说明：xovsprs 在当前种群 OldChrom 一对个体间按交叉概率 XOVR 进行减少代理的单点交叉，并返回交配后的新种群 NewChrom。在 OldChrom 和 NewChrom 中每一行对应一个体。它适用于任意染色体表示。

XOVR 是一可选参数，指明交叉概率，如果省略、空或为 NaN，则设 XOVR=0.7。

交配的对是有序的，即奇行与它下一个偶行配对。如果矩阵 OldChrom 行数是奇数行，则最后一行不参加交配，因此，种群将按交配要求组织成连续的对。这可使用函数 ranking 计算每个染色体的适应度，并由选择函数(select，sus 或 rws)在种群中用与适应度相关的概率选择个体来完成。

xovsprs 是一低级交叉函数，通常被函数 recombin 调用。

例 **5.27** 减少代理的单点交叉函数 xovsprs 的应用举例。下面的程序代码为 2 个个体种群的减少代理的单点交叉示例(交叉概率为 0.7)：

```
Chrom=crtbp(2，8)；   % 创建 2 个个体的种群
```

$$Chrom=\begin{bmatrix}1 & 1 & 0 & 0 & 0 & 0 & 1 & 1\\0 & 1 & 0 & 1 & 1 & 1 & 0 & 1\end{bmatrix}$$

```
NewChrom=xovsprs(Chrom，0.7)；   % 交叉概率为 0.7
```

则 Chrom 成为如下形式：

$$NewChrom=\begin{bmatrix}1 & 1 & 0 & 0 & 1 & 1 & 0 & 1\\0 & 1 & 0 & 1 & 0 & 0 & 1 & 1\end{bmatrix}$$

算法说明：单点交叉参见 xovsp。

交叉对任何可能点总是产生新个体，减少代理算子限制它。它是通过限制交叉点的位置来完成的，例如，交叉点只能发生在基因不同的地方。

xovsprs 使用适当的参数调用 xovmp。

参见：xovsp，xovdp，xovdprs，xovsh，xovshrs，xovmp，recombin，select。

# 第六章　遗传算法工具箱的应用

本章主要介绍英国设菲尔德大学开发的遗传算法工具箱的使用方法。这个遗传算法工具箱已经在世界上近 30 个广泛的应用领域得到了很好的测试，包括参数优化、多目标优化、控制器结构选择、非线性系统论证、形形色色模式的模型制作、神经网络设计、实时和自适应控制、并行遗传算法、故障诊断和天线设计等。

## 6.1　安　　装

遗传算法工具箱的安装过程可以参照 MATLAB 的安装说明进行。建议将工具箱中的文件保存在 MATLAB 或工具箱主目录下名为 genetic 的子目录中。

工具箱包含许多可用的示例。例如，示例 M 文件 sga.m 是解决数值优化问题的单种群二进制编码的遗传算法。示例 M 文件 mpga.m 是解决动态控制问题的多种群十进制实数编码的遗传算法。这两个示例 M 文件将在第七章进行详细的讨论。

另外，来自遗传算法著作的一套测试函数在一个从遗传算法工具箱函数中分离出来的目录 test-fns 中提供，这些测试函数的摘要性描述在最后给出。此外，还有一个文档描述了这些函数的实现和使用。

## 6.2　种群的表示和初始化

遗传算法同时运算在由已编码的参数集组成的称为种群的大量可能答案上。典型的一个种群由 30～100 个个体组成，一些变化的称为微型的遗传算法采用很小的 10 个个体的种群，使用有限制的复制和代替策略以试图达到实时运算。

一般情况下，在遗传算法中大多数采用单层的二进制串染色体表示方法。这里，参数集中的每一个决策变量被编码为一个二进制串，它串接起来形成一个染色体。这里采用格雷码，可用来克服传统的二进制表示方法的不足。Caruana 和 Schaffer 的经验证明显示，表示图中邻近值间过大的海明(Hamming)距离使用标准的二进制表示情况下，搜索过程易导致欺骗性结果或不能有效定位于全局最小值。Schmitendorgf 等更进一步的方法是在二进制编码的染色体变换到实际的表示值时使用对数计量。尽管参数值的精度可能低于希望的范围，但在问题中，可行参数的传播是未知的，一个大的搜索空间中可能掩藏着大量相同的位，与线性映像方案允许的搜索未知搜索空间的计算负担相比，可减少到更易管理的水平。

与此同时，二进制编码的遗传算法使用更广泛，同时引起更多兴趣的可选择的编码方案，有整数和实数表示。对一些问题域，有一个争论是二进制表示事实上是靠不住的，它

掩盖了搜索的自然性。例如在选择子集问题中，使用整数表示法为搜索表提供方便和自然表示方法映射为对问题域的表示。

Wright 声称使用实值基因的遗传算法在数值函数优化上与二进制编码相比有许多的优点。在函数计算前，不需要从染色体到表现值的转换，提高了遗传算法的效率；计算机内部高效的浮点表示可直接使用，减少了内存需求；相对于离散的二进制或其他值，没有精度损失；对使用不同的遗传算子非常自由。Michalewicz 在其著作《进化策略(Evolution Strategies)》中描述了使用实值编码的细节。

有了确定的表示，简单遗传算法的第一步是建立初始种群，使用在希望的范围内均匀分布数字的随机数产生器生成所需数量的个体。例如，使用有 $N_{ind}$ 个个体的二进制种群，染色体长度为 $L_{ind}$ 位，从集{0，1}中产生 $N_{ind} \times L_{ind}$ 个均匀分布随机数。一个变化是 Bramlette 的扩展随机的初始步骤，无论如何，大量的随机初始化(Initializations)为每个个体尝试，它们中性能最好的被选择为初始种群，另外一些遗传算法用户使用初始种群的一些已知为接近全局最小化的个体播种。当然，这种接近法仅仅适用于自然问题，是易于事前了解的或遗传算法用于与基本知识系统相联合的。

这个遗传算法工具箱支持二进制、整数和浮点的染色体表示。二进制和整数种群初始化可使用工具箱函数 crtbp 创建二进制种群。附加函数 crtbase 提供向量描述的整数表示法。实值种群的初始化可使用函数 crtrp 完成，程序 bs2rv 提供二进制串和实值之间的转换，它支持使用格雷码和对数编码。

## 6.3 目标函数和适应度函数

目标函数将提供一测量手段，测量个体在问题域的完成情况。在一个最小化问题中，最适合的个体对应最小的目标函数值。未经加工的适应度值通常只用在遗传算法的中期来判断一个个体的相对性能。另一函数——适应度函数通常用于转换目标函数值为相对适应度值。因此，有

$$F(x) = gf(x)$$

这里 $f$ 是目标函数，$g$ 是将目标函数值转换为非负值的变换因子，$F$ 是所得的相对适应度，当目标函数是最小化即函数值越小对应适应度越好的个体时，这种变换是必需的。许多情况下，适应度函数值对应大量子代——预计在下一代中能存在的个体。通常使用这个转换进行适应度概率分配。个体的适应度——每一个个体的 $F(x_i)$ 通过个体的未加工的适应度 $f(x_i)$ 相对整个种群的适应度被计算出来，即

$$F(x_i) = \frac{f(x_i)}{\sum_{i=1}^{N_{ind}} f(x_i)} \tag{6.1}$$

式中，$N_{ind}$ 是种群大小，$x_i$ 是个体 $i$ 的表现值，与此同时适应度分配确保每一个体均有按其相对适应度再生的机会，它不能处理负的目标函数值。

使用水平移位的线性变换是优先使用的适应度分配方法。

$$F(x) = a \cdot f(x) + b \tag{6.2}$$

如果优化方法是最大化，这里 $a$ 是一个正的换算系数；如果是最小化，则 $a$ 是一个负值。$b$ 是一个偏移值，它确保最后的适应度值是非负的。

无论如何，上面给出的线性尺度变换和移位方法易迅速收敛。这个选择算法选择个体进行再生是基于它们的相对适应度的。使用线性尺度变换，预期的后代将近似与它们性能相称(成比率)，如果说没有限制个体在给出代中的性能，则在上代中高适应度的个体将决定再生过程引起快速收敛，可能产生局部最优解。同样，如果种群有很少变异，这种变换仅仅向最合适的个体提供小偏移。

Baker 认为通过限制再生范围，以使个体不产生极端的后代，防止过早收敛。这里个体是根据它们在种群中的排序计算适应度。一个变量 $MAX$ 常常用来决定偏移或选择强度，最适合的个体和其他个体的适应度由下面的规则决定：

$$MIN=2.0-MAX$$

$$INC=2.0\times(MAX-1.0)/N_{ind}$$

$$LOW=INC/2.0$$

这里，$MIN$ 是下界，$INC$ 是邻近个体适应度的差别，$LOW$ 是最小适应度个体预期的试验值(一定选择代数)，$MAX$ 是在区间[1.1，2.0]中的典型精选值。因此，如果种群大小 $N_{ind}$ 为 40，$MAX=1.1$，我们将获得 $MIN=0.9$，$INC=0.05$，$LOW=0.025$。种群中个体的适应度能被直接计算出：

$$F(x_i)=2-MAX+2(MAX-1)\frac{x_i-1}{N_{ind}-1} \qquad (6.3)$$

式中，$x_i$ 是个体 $i$ 在有序种群中的位置。

尽管工具箱提供大量的 M 文件例子，用于完成通用的测试功能，但目标函数必须由用户创建。这些目标函数都有文件名前缀 obj，这个工具箱支持线性和非线性两种评定法，包括一个简单的线性尺度变换函数 scaling，较为完备。需要注意的是，线性尺度变换函数不适合于目标函数产生负的适应度值的情况。

## 6.4 选　　择

选择是由遗传代数或试验值决定的过程。一个特殊的个体为再生被挑选，确定编码子代中的一个个体被产生，这种个体的选择可以看做两个分离的过程：

(1) 决定试验的代数和希望接收的个体。

(2) 转换预期的试验数为大量离散的子代。

第一部分所关心的是转换原始的适应度值为个体再生机率的预期的实值和处理使用前一小部分的适应度计算。第二部分是基于一个个体相对另一个体的适应度为再生进行的个体概率选择，有时是由抽样而已知的。这一小部分的剩余部分将回顾现行使用的更为流行的选择方法。

Baker 为选择算法展现了性能的三种措施——偏差(Bias)、个体扩展(Spread)、效率(Efficiency)。Bias 被定义为个体的实际值与预期的选择概率之间的绝对差值。因此当个体的选择概率等于它的预期试验值时，获得最优零偏差。Spread 是个体可能达到的一个可能实验子代数的范围，如果 $f(i)$是个体 $i$ 收到的实际实验代数，则最小个体扩展就是最小的

Spread，即理论上允许零偏差。

$$f(i) \in \{\lfloor et(i) \rfloor, \lceil et(i) \rceil\}$$

这里，$et(i)$是个体 $i$ 的预期实验代数，$\lfloor et(i) \rfloor$表示下限，$\lceil et(i) \rceil$表示上限。因此，当 Bias 是一个精确的指示时，可选择方法的个体扩展来测量它的一致性(Consistency)。

获得有效选择方法的要求是由必须保持遗传算法全面的时间复杂度决定的。在一些著作中显示出遗传算法的另一些方面(除实际的目标函数评估)，即 $O(L_{ind} \times N_{ind})$或更好的时间复杂度，这里 $L_{ind}$是个体长度，$N_{ind}$是种群的大小。由这个选择算法而得到零偏差，与此同时保持了最小个体扩展，但对于改进这个遗传算法的时间复杂度不会有任何作用。下面介绍两种选择方法。

**1. 轮盘赌选择算法**

许多的选择技术采用轮盘赌原理，个体的选择概率是基于它们的性能进行的一些计算。实值范围——总和是所有个体期望的选择概率的总和或当前种群中所有个体原始适应度值的总和。个体采用一对一方式映像到范围[0，$Sum$]的一连续区间，每一个体区间的大小与对应个体的适应度值相匹配。如图 6.1 所示，轮盘赌轮的周长是 6 个个体适应度值的总和，个体 5 是最大适应度个体，它占有最大的区间，而个体 6 和 4 是最小适应度个体，相对应地在轮盘上占有小的区间。选择一个个体，用在[0，$Sum$]间产生随机数，看此随机数处在哪个个体的区间上，则此个体被选中。重复此过程，直到所需数量个体被选中为止。

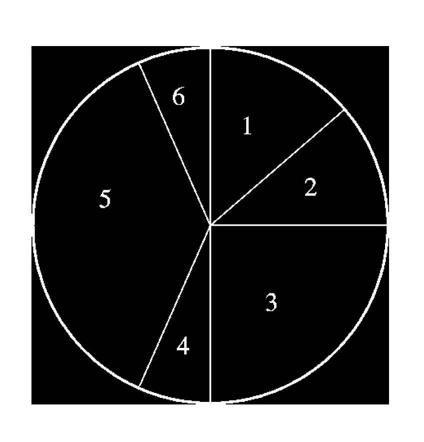



图 6.1 轮盘赌选择

这个基本的轮盘赌选择方法是可代替的随机抽样(SSR)。在这里片段大小和选择概率在整个选择阶段保持相同，个体的选择根据上面的步骤完成。SSR 给出了零偏差但可能是无限的个体扩展，段长大于零的任意个体完全可能填入下一种群。

局部替换的随机抽样(SSPR)是在 SSR 上扩充而来的，如果一个个体被选择，则重新建立它的段大小。个体被选择一次，其段长被减小 1，如果段长变为负，则设为零。这里提供一个体扩展上界$\lceil et(i) \rceil$，无论如何低界为零，偏差(Bias)都是大于 SSR 的。

剩余抽样法解决了两个不同方面的问题。

(1) 在整数方面，个体的选择决定于期望试验的整数部分，剩余个体的选择概率来自于个体们期望值的小数部分。替换的剩余随机抽样(RSSR)采用轮盘赌选择，对抽样的个体并不重新赋值。

(2) 在轮盘赌选择期间，个体的小数部分保持不变，随后在“Spins”间竞争选择。RSSR 提供零偏差和具有下界的个体扩展，上界被指定代的参与分配样本部分和一个个体的整数部分大小限定。例如，任何一个小数部分大于零的个体将在分级(小数)阶段的所有抽样中取胜。如果一个个体在小数阶段被抽样，没有替换的剩余随机抽样(RSSWR)设置它的期望值的小数部分为零。这里给出的 RSSWR 具有最小的个体扩展。

**2. 随机遍历抽样**

随机遍历抽样(SUS)是具有零偏差和最小个体扩展的单状态(single_phase)抽样算法。替代用于轮盘赌方法的单个选择指针，SUS 使用 $N$ 个相等距离的指针，这里 $N$ 是要求选择的个数。种群被随机排列，在[0，$Sum/N$]范围内产生一随机数作为指针 $ptr$，$N$ 个个体

由相隔一个距离的 $N$ 个指针[$ptr$，$ptr+1$，$ptr+2$，…，$ptr+N-1$]选择，选取适应度范围在指针位置的个体。一个个体确保被选择的最少次数[$et(i)$]和不超过[$et(i)$]，因此获得最小的个体扩展，另外个体完全按它们在种群中的地位选择，SUS 具有零偏差。

轮盘赌选择算法总可按 $O(N\lg N)$执行，而 SUS 是更简单的算法且具有 $O(N)$的时间复杂度。这个工具箱提供随机遍历抽样函数 sus 和具有替换的随机抽样算法 rws。

## 6.5 交　　叉

在遗传算法中产生新染色体的基本操作是染色体的交叉(也称为基因重组)。与自然进化一样，交叉产生的新个体具有父母双方的一部分遗传物质。最简单的交叉是单点交叉。下面讨论几种交叉方法，并比较它们各自的优点。

**1.** 多点交叉

对多点交叉有 $m$ 个交叉位(特别当 $m=1$ 时为单点交叉)，$k_i\in\{1,2,\cdots,l-1\}$，这里 $k_i$ 是交叉点，$l$ 是染色体的长度，这 $m$ 个交叉位是通过随机数选择的、没有重复的、按升序排列的。父母染色体中在两个相连的交叉位间的基因进行交换，产生两个新的子代。个体第一个基因到第一个交叉点之间的位并不进行交换。这个过程如图 6.2 所示。

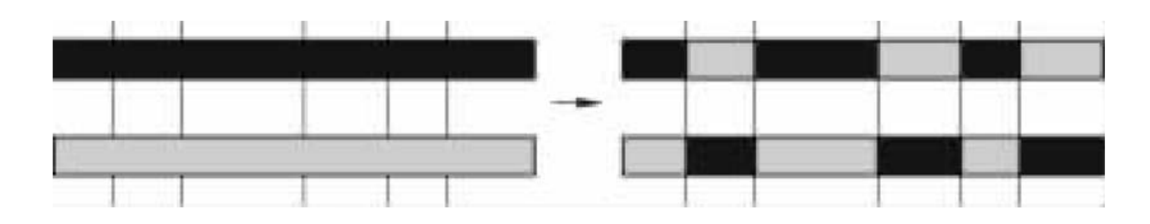


图 6.2　多点交叉($m=5$)

多点交叉的思想和事实交叉上的许多变异源于控制个体特定行为的染色体表示信息的部分，无须包含于邻近的子串中。进一步，多点交叉的破坏性可以促进解空间的搜索而不至于在搜索中因高适应度个体过早收敛，使搜索更加健壮。

**2.** 均匀交叉

单点交叉和多点交叉定义的交叉点即染色体拆分基因位。均匀交叉更加广义化，将每个点都作为潜在的交叉点。一个交叉掩码，随机产生的与个体等长的染色体结构，掩码中的等价位表明哪个父个体向子个体提供变量值。考虑如下两个父个体，交叉掩码和最终的子个体：

$P_1=1\ 0\ 1\ 1\ 0\ 0\ 0\ 1\ 1\ 1$

$P_2=0\ 0\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 0$

$Mask=0\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 0\ 0$

$O_1=0\ 0\ 1\ 1\ 1\ 1\ 0\ 1\ 0\ 0$

$O_2=1\ 0\ 0\ 1\ 0\ 0\ 1\ 0\ 1\ 1$

这里，第一个子个体 $O_1$ 产生的位，其掩码对应位为 1，则来自父个体 $P_1$；对应位为 0，则来自父个体 $P_2$。而第二个个体的产生则相反，即将上面的 $P_1$ 和 $P_2$ 交换。

均匀交叉类似于多点交叉，可以减少二进制编码长度与给定参数特殊编码之间的偏

差，并有助于克服单点交叉中对短子串的偏差，而不用精确了解染色体表示信息的个体位的重要性(意义)。Spears 和 De Jong 已经证明均匀交叉通过应用概率交换位怎样参数化。额外的参数常用来控制重组期间的破坏总量，不用引进表现信息长度用的偏差。当均匀交叉用于实值型时，它常作为离散重组参考。

**3.** 中间重组

给定一实值编码的染色体结构，中间重组是产生新表现型范围的方法，这个范围处于父表现型值之间。子个体按照下列公式产生：

$$Q_1 = \alpha P_1 \cdot (P_2 - P_1) \tag{6.4}$$

这里，$\alpha$ 是一区间的均匀随机数，是换算系数，典型区间为[-0.25，1.25]；$P_1$ 和 $P_2$ 是父染色体，子代的每一个变量的值由父变量的值和对每一对父基因产生的一新的 $\alpha$ 值按上面的公式产生。用几何图形描述，中间重组能产生新的变量在比其父代定义的稍大的立体中，并受 $\alpha$ 限制，显示如图 6.3 所示。

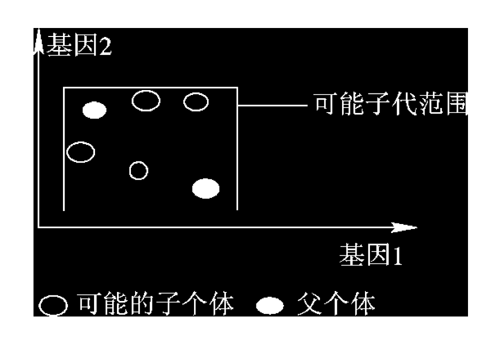



图 6.3　中间重组的几何效果

**4.** 线性重组

线性重组与中间重组比较相似，只是在重组中使用一个 $\alpha$ 值。图 6.4 显示了线性重组怎样由父个体在 $\alpha$ 扰动范围的线上任意点产生，用于两个参数的重组。

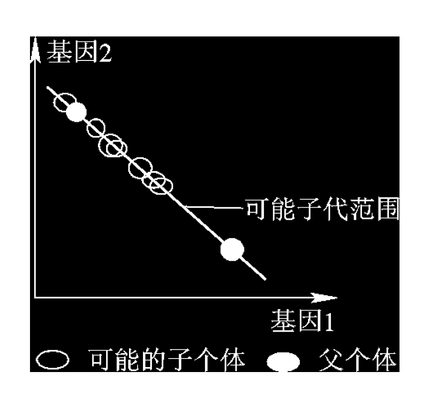



图 6.4　线性重组的几何效果

**5.** 其他一些交叉算子

一相关的交叉算法是洗牌交叉。一个交叉点被选择，但此位前的哪些位交换，这些位在父母之间进行随机洗牌。重组后，子代中的这些位不进行洗牌，当这些位在每次交叉执行时被随机分配，将减少位置偏差。

在任何可能的地方，缩小代理算子、强制交叉总是产生新的个体群。通常它作为限制交叉点位置的工具，以便使交叉点只发生在那些基因值不同的地方。

## 6.6 变　　异

在自然进化中，变异是一随机过程，是基因上的一个等位基因被另一个代替而产生一新的遗传结构。在遗传算法中，变异采用了一任意的小概率，典型值是在 0.001～0.1 之间，并改变染色体的元素。通常认为它是后台算子，变异的作用被认为是：搜索任意给定串的可能性永不为零，为保证通过选择和交叉操作恢复可能丢失的好的遗传物质提供安全网络。

图 6.5 说明了在一个二进制串上的变异效果。二进制串是用 10 位二进制染色体表示的一个在区间[0，10]之间的实值编码，使用标准二进制和格雷码两种编码，二进制串中的变异位是第 3 位。这里，二进制变异是对选择变异位的值进行翻转。给定的变异通常一致应用于整个种群串，一个给定的二进制串变异的可能不止一位。

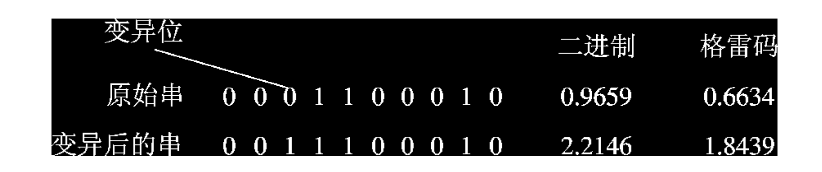



图 6.5　二进制变异

对于非二进制表示，变异完成的是扰乱基因值和随机选择一允许范围内的新值。Wright、Janikow 和 Michalewicz 已经证明，实值编码在高变异率中比二进制编码好，它提高了对搜索空间的搜索能力，而不会对收敛特征产生不利影响。的确，Tate 和 Smith 论证了为什么比二进制复杂的编码，高变异率是适合的和必需的，示范了为什么复杂组合最优化问题、高变异率和非二进制编码领域解法大大好于通用方法。

还有许多变异的变异算子，例如，具有低适应度的、带偏差的变异能提高搜索能力，而不会丢失来自高适应度个体的信息，或带参数的变异的变异概率降低不影响种群收敛。Mühlenbein 介绍了一种实值编码遗传算法的变异算子，对变异范围分布用非线性项应用于基因值。已经证明使用偏差变异使基因值有较小的变化，变异常常用于与重组连接作为搜索过程的前期工作。另一些变异包括传统变异如何在染色体中分配个体基因，习惯上对弱条件直接变异和重新安排变异，即交换位的位置或提高在决定变量空间基因的差异。

遗传算法工具箱提供的二进制和整型变异函数是 mut，实值变异函数是 mutbga；变异算子的高级接口函数是 mutate。

## 6.7 重　插　入

一旦一个新的种群通过对旧种群的个体进行选择和重组而产生，新种群中个体的适应度被确定了，如果通过重组产生的种群个体数少于原始种群的大小，新种群和旧种群大小的差异被称为代沟。在这种情况下，每一代产生的新个体数较少，这时的遗传算法称之为稳态或增量的。如果一个或多个最适合个体被连续代繁殖，则遗传算法被称为得到精英策略。

为了保持原始种群的大小，一些新的个体不得不被重新插入旧种群中。同样地，如果

在每一代并非所有新个体被使用或生产的子代大小超过旧种群，则一个恢复计划用来决定那些个体存在于新种群中。如前所述，稳态遗传算法(Steady-State GA)最引人注目的一个重要特征是在每一代中不创建比现存种群多的后代，计算次数被减少，并且产生后代时由于有少的新个体存储，内存要求也小。

当选择老种群中哪些成员被替代时，最常用的策略是替换确定的最小适应度成员。Fogarty 在研究中已经证明，对选择的个体进行替代使用逆比率选择或最小适应度，其收敛特性没有显著差异。当最大适应度成员将具有机会连续生存时，他进一步断言替代最小适应成员是一有效的优选策略工具。的确，大多数成功的选择方案是选择种群中最老的成员进行替代。这里指出多数与代的再生一致，在某一时刻，种群中的每一个成员都将被替代。因此为使一个个体能连续存在，它必须有足够适应度以确保繁殖到下一代。

遗传算法工具箱提供一函数 reins 使个体可在重组后重插入种群中。任选输入参数允许使用一致随机或基于适应度的重插入。另外，这个程序能选择重插入比重组产生少的子代。

## 6.8 遗传算法的终止

因为遗传算法是随机搜索算法，找到一个正式的、明确的收敛性判别标准是困难的。若在找到最优个体以前的许多代的种群适应度可能保持不变，应用程序的常规终止条件将成为有问题的。常用方法是遗传算法终止采用达到预先设定的代数和根据问题定义测试种群中最优个体的性能。如果没有可接受的解答，遗传算法重新启动或用新的搜索初始条件。

## 6.9 数据结构

MATLAB 本质上只支持实值或复数值矩阵一种数据类型。遗传算法工具箱的主要数据结构是染色体、表现型、目标函数值和适应度值。

**1. 染色体**

染色体数据结构用大小 $N_{ind} \times L_{ind}$ 的单矩阵存储整个种群，这里 $N_{ind}$ 是种群中个体的个数，$L_{ind}$ 是个体基因表现型的长度，每一行对应一个个体的基因，由 $n$ 个基数组成，典型的是二进制值。

染色体数据结构举例如下：

$$\text{Chrom} = \begin{bmatrix} g_{1,1} & g_{1,2} & \cdots & g_{1,L_{ind}} \\ g_{2,1} & g_{2,2} & \cdots & g_{2,L_{ind}} \\ \vdots & \vdots & & \vdots \\ g_{N_{ind},1} & g_{N_{ind},2} & \cdots & g_{N_{ind},L_{ind}} \end{bmatrix} \begin{matrix} \% \text{ 个体 } 1 \\ \% \text{ 个体 } 2 \\ \vdots \\ \% \text{ 个体 } N_{ind} \end{matrix}$$

这种数据表示并不是染色体结构的强制结构，只要求所有染色体具有相同的长度。因此，结构化种群或具有可变基因基础的种群可用于遗传算法工具箱提供的合适编码函数、映射染色体的表现型。

**2．**表现型

遗传算法中的决策变量或表现型通过在决策变量空间对染色体表现形式进行映射获得。这里，包含在染色体结构中的每一个串根据在搜索空间中的维度值和对应的决策变量向量值编码为一行向量 $N_{var}$，这个决策变量被存储在一大小为 $N_{ind} \times N_{var}$ 的数字矩阵中，每一行对应一特定个体的表现型。下面给出了表现型数据结构的例子，遗传算法工具箱的 DECODE 常用于表示一泛函数，映射基因到表现型。

Phen ＝DECODE（ Chrom ）；％ 将基因型变换为显型(Map Genotype to Phenotype)

$$= \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,N_{var}} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,N_{var}} \\ \vdots & \vdots & & \vdots \\ x_{N_{ind},1} & x_{N_{ind},2} & \cdots & x_{N_{ind},N_{var}} \end{bmatrix} \begin{matrix} \%\ 个体\ 1 \\ \%\ 个体\ 2 \\ \vdots \\ \%\ 个体\ N_{ind} \end{matrix}$$

在染色体表示和表现型值之间的实际映射依赖于函数 DECODE 的应用。应用这种表达得到不同类型决策变量的向量是完全可行的。例如，在相同的表现型数据结构中混有整型、实型和字母的决策变量是允许的。

**3．**目标函数值

目标函数常用来评估表现型在问题域中的性能。目标函数值可能是标量或在多目标情况下是矢量。值得注意的是，目标函数值不像适应度值一样是必需的。

目标函数值被存储在一大小为 $N_{ind} \times N_{obj}$ 的数字矩阵中，这里 $N_{obj}$ 是目标的数量。每一行对应一单独个体的目标矢量。目标函数值数据结构例子在下面给出，用 OBJFUN 表示一任意的目标函数。

ObjV ＝OBJFUN (Phen)；　％ 目标函数(Objective Function)

$$= \begin{bmatrix} y_{1,1} & y_{1,2} & \cdots & y_{1,N_{var}} \\ y_{2,1} & y_{2,2} & \cdots & y_{2,N_{var}} \\ \vdots & \vdots & & \vdots \\ y_{N_{ind},1} & y_{N_{ind},2} & \cdots & y_{N_{ind},N_{var}} \end{bmatrix} \begin{matrix} \%\ 个体\ 1 \\ \%\ 个体\ 2 \\ \vdots \\ \%\ 个体\ N_{ind} \end{matrix}$$

**4．**适应度值

适应度值是由目标函数值通过计算或评定等级而得出的。适应度是一非负的标量并保存在长度为 $N_{ind}$ 的列向量中。下面给出一个例子。FITNESS 是一任意的适应度函数。

Fitn ＝FITNESS(ObjV)；　％ 适应度函数(Fitness Function)

$$= \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_{N_{ind}} \end{bmatrix} \begin{matrix} \%\ 个体\ 1 \\ \%\ 个体\ 2 \\ \vdots \\ \%\ 个体\ N_{ind} \end{matrix}$$

注意，对多目标函数，一个特定个体的适应度是目标函数值向量的一个函数。多目标问题一般没有一个简单的惟一解，但可由具有不同值的决策变量的一组相同合适解描述。因此通过采用一些措施确保种群能够解出一组 Pareto 最优解，例如，通过在选择方法中使

用适应度共享。尽管在本版本的遗传算法工具箱中不支持，计划在后续版本中补充多目标搜索。

## 6.10 多种群支持

遗传算法工具箱通过使用高级遗传算子函数和例程在子种群中交换个体而支持多子种群。在一些文献中，使用多子种群表明，在大多数情况下，相比于单种群遗传算法，它改善了所获得结果的性能。

遗传算法工具箱支持通过修改所用的数据结构将所用单种群分割成许多子种群或同类群，以便使用一个简单矩阵将子种群保存在一连续块中。例如，染色体结构 Chrom 由每个长度为 $N$ 的 $SUBPOP$ 个子种群组成而保存。

著名的是迁移(Migration)或孤岛 (Island)模型，每一个子种群通过传统的遗传算法和个体时，随时从一个子种群迁移到另一个子种群而发展成下一代。迁移的个体数量和迁移的模式决定有多大的遗传差异发生。

为了允许工具箱例程在子种群上独立操作，遗传算法工具箱提供了大量的高级接口程序，接受一任选变量来确定在数据结构中获得的子种群的数量。低级例程被依次独立调用，对每个子种群完成选择、交叉和重插入等功能。表 6.1 列出了子种群支持的函数，这是一些高级函数。

表 **6.1** 子种群支持的函数

| 函数 | 说明 |
|---|---|
| mutate | 变异算子 |
| recombin | 交叉和重组算子 |
| reins | 均匀随机和基于适应度的重插入 |
| select | 独立子种群选择 |

注意：对当前工具，所有子种群必须是相同大小的。

个体在两个子种群之间的迁移是由函数 migrate 实现的，一个单一的标量决定迁移个体的总量。因此，给出的一个种群包括大量的子种群，它从另一个子种群接收数量相同的个体，这个函数的第二个参数控制使用均匀选择或者按照适应度两种方式之一选择哪个个体参加迁移。均匀选择为迁移选取个体，并且在子种群中使用随机方式用迁移个体替代本子种群中的个体，基于适应度迁移选择个体是按照它们的适应度水平进行的，最适应个体将被选择作为迁移，子种群被替代的个体是按均匀随机选择的。

下面进一步用参数说明在不同种群拓扑结构中迁移的发生过程。图 6.6 显示循环迁移在函数 migrate 中执行的大多数基本迁移路径。这里个体的迁移是在直接相邻的子种群中，例如来自子种群 6 的迁移个体只迁移到子种群 1，而来自子种群 1 的个体只迁移到子种群 2。

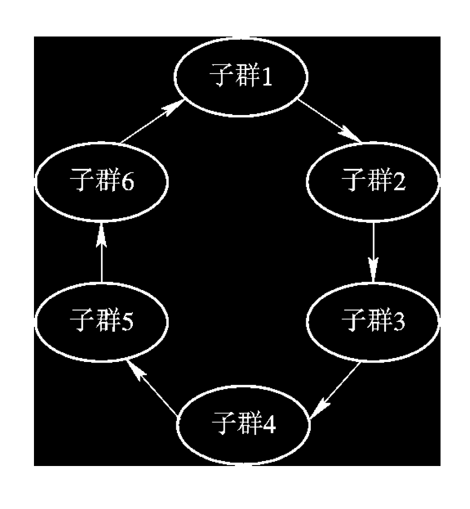



图 6.6　循环迁移拓扑

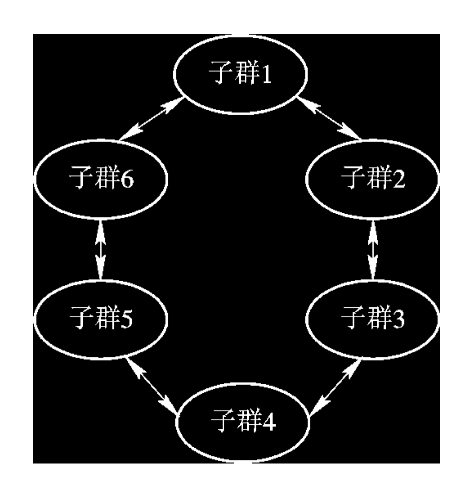



图 6.7　相邻迁移拓扑

循环迁移的相似策略是相邻迁移，如图 6.7 所示。像循环拓扑，迁移只发生在最近的邻居之间，但可发生在相邻子种群之间的任意方向。对每一个子种群迁移个体根据要求的选择方法决定，它来自于相邻子种群和从个体池中最后选择的，确保个体不会从一个子种群迁移到相同子种群。

函数 migrate 所支持的是最通常的迁移，是没有限制的迁移，如图 6.8 所示。这里个体可从任意一个个体迁移到另一个个体。对每一个子群，有一群来自其他子群的潜在的迁移者，这些个体迁移者按照适合的选择策略决定。

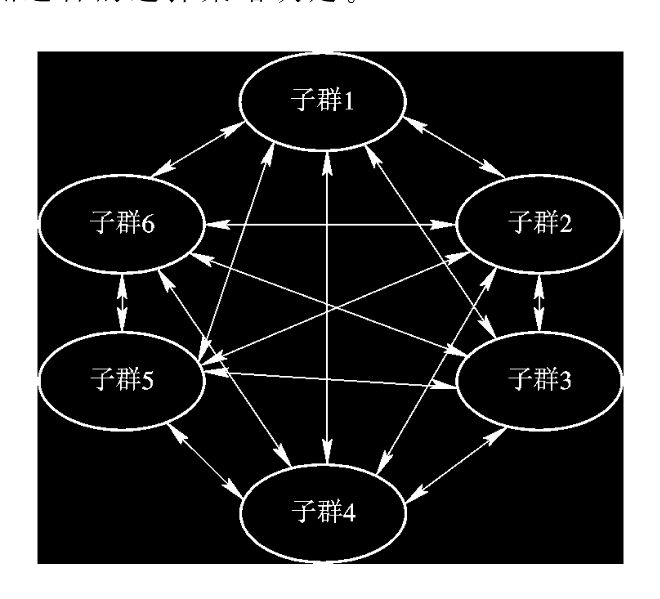



图 6.8　自由迁移拓扑

# 6.11　示范脚本

遗传算法工具箱提供的 GA 脚本文件完成了大量测试函数，这些测试函数被提供在一个独立目录 test_fns 中，包括主要的示范脚本和工具箱例程，伴随的附录文件放在 test_fns.ps 目录中，给出了问题执行的完全细节。表 6.2 总结了工具箱提供的测试文件。

表 6.2　工具箱提供的测试文件

| 序号 | M 文件名 | 说　　明 |
| --- | --- | --- |
| 1 | objfun1 | De Jong's 函数 1 |
| 2 | objfun1a | axis parallel hyper-ellipsoid(轴并行超球体) |
| 3 | objfun1b | rotated hyper-ellipsoid (旋转超球体) |
| 4 | objfun2 | Rosenbrock's valley (banana function) |
| 5 | objfun6 | Rastrigin's function |
| 6 | objfun7 | Schwefel's function |
| 7 | objfun8 | Griewangk's function |
| 8 | objfun9 | sum of different powers (不同权的总和) |
| 9 | objdopi | double integrator (双积分) |
| 10 | objharv | harvest problem (收获问题) |
| 11 | objlinq | discrete linear-quadratic problem (离散二次线性问题) |
| 12 | objlinq2 | continuous linear-quadratic problem (连续二次线性问题) |
| 13 | objpush | push-cart problem (装载问题) |

# 第七章　遗传算法应用举例

遗传算法提供了一种求解非线性、多模型、多目标等复杂系统优化问题的通用框架，它不依赖于问题所属的具体领域。随着对遗传算法技术的不断研究，人们对遗传算法的实际应用也越来越重视，它已经广泛地应用于函数优化、组合优化、自动控制、机器人学、图像处理、人工生命、遗传编码、机器学习等科技领域。遗传算法已经在求解旅行商问题、背包问题、装箱问题、图形划分问题等方面取得了成功。本章通过一些具体的例子，介绍如何利用第五章提供的遗传算法通用函数编写 MATLAB 程序，解决实际问题。

## 7.1　简单一元函数优化实例

利用遗传算法计算下面函数的最大值：

$$f(x) = x\sin(10\pi \cdot x) + 2.0, \quad x \in [-1, 2]$$

选择二进制编码，种群中个体数目为 40，每个种群的长度为 20，使用代沟为 0.9，最大遗传代数为 25。

下面为一元函数优化问题的 MATLAB 代码。

```
Figure(1);
fplot ('variable. * sin (10 * pi * variable)+2.0',[-1,2]);   % 画出函数曲线
% 定义遗传算法参数
NIND=40;            % 个体数目(Number of individuals)
MAXGEN=25;          % 最大遗传代数(Maximum number. of generations)
PRECI=20;           % 变量的二进制位数(Precision of variables)
GGAP=0.9;           % 代沟(Generation gap)
trace=zeros (2, MAXGEN);                     % 寻优结果的初始值
FieldD=[20;-1;2;1;0;1;1];                    % 区域描述器(Build field descriptor)
Chrom=crtbp(NIND, PRECI);                    % 初始种群
gen=0;                                       % 代计数器
variable=bs2rv(Chrom,FieldD);                % 计算初始种群的十进制转换
ObjV=variable. * sin (10 * pi * variable)+2.0;   % 计算目标函数值
while gen<MAXGEN,
    FitnV=ranking(-ObjV);                    % 分配适应度值(Assign fitness values)
    SelCh=select('sus', Chrom, FitnV, GGAP); % 选择
    SelCh=recombin ('xovsp',SelCh,0.7);      % 重组
    SelCh=mut(SelCh);                        % 变异
    variable=bs2rv(SelCh,FieldD);            % 子代个体的十进制转换
```

```
        ObjVSel=variable. * sin(10 * pi * variable)+2.0;               % 计算子代的目标函数值
        [Chrom ObjV]=reins(Chrom,SelCh,1,1,ObjV,ObjVSel);       % 重插入子代的新种群
        gen=gen+1;                                                          % 代计数器增加
        % 输出最优解及其序号，并在目标函数图像中标出，Y为最优解，I为种群的序号
        [Y,I]=max(ObjV), hold on;
        plot (variable (I),Y, 'bo');
        trace (1,gen)=max (ObjV);                                          %遗传算法性能跟踪
        trace (2,gen)=sum (ObjV)/length (ObjV);
    end
    variable=bs2rv (Chrom,FieldD);                                          %最优个体的十进制转换
    hold on, grid;
    plot (variable',ObjV','b * ');
    Figure (2);
    plot (trace (1,:)');
    hold on;
    plot (trace (2,:)','-.');grid;
    legend ('解的变化','种群均值的变化').
```

使用基于适应度的重插入确保四个最适应的个体总是被连续传播到下一代。这样，在每一代中有36(NIND * GGAP)个新个体产生。

区域描述器FieldD描述染色体的表示和解释，每个格雷码采用20位二进制，变量区间为[-1，2]。

程序段Chrom=crtbp (NIND，PRECI)表示一个初始种群Chrom被函数crtbp创建，它是由NIND个均匀分布、长度为PRECI的二进制串构成的。

基于排序的适应度分配计算由程序段FitnV=ranking (-ObjV)实现。

对这个等级评定算法的缺省设置是选择等差为2和使用线性评估，给最适应个体的适应度值为2，最差个体的适应度值为0，这里的评定算法假设目标函数是最小化的，所以ObjV前加了一个负号，使目标函数最大化。适应度值结果由向量FitnV返回。

选择层使用高级函数选择调用低级函数随机遍历抽样例程sus，SelCh包含来自原始染色体的GGAP * NIND个个体，这些个体将使用高级函数recombin进行重组，recombin使个体通过SelCh被选择再生产，并使用单点交叉例程xovsp，使用交叉概率Px=0.7执行并叉。交叉后产生的子代被同一个矩阵SelCh返回，实际使用的交叉例程通过支持使用不同函数名字串传递给recombin而改变。

为了产生一组子代，变异使用变异函数mut。子代再次由矩阵SelCh返回，变异概率缺省值PM=0.7/Lind=0.0017，这里Lind是假定的个体长度。再次使用bs2rv，将个体的二进制编码转化为十进制编码，计算子代的目标函数值ObjVSel。

由于使用了代沟，子代的数量比当前种群数量要小，因此使用恢复函数reins。这里Chrom和SelCh是矩阵，包含原始种群和子代结果。这两个事件的第一个被使用单个种群和采用基于适应度的恢复，基于适应度的恢复用SelCh中的个体代替Chrom中最不适应的个体。原始种群中个体的目标函数值ObjV随后又作为函数reins的输入参数，子代中个体的目标函数值由ObjVSel提供。reins返回具有插入子代的新种群Chrom和该种群中个

体的目标函数值 ObjV。

每次迭代后的最优解和解的均值存放在 trace 中。这个遗传优化的结果包含在矩阵 ObjV 中。决策变量的值为 variable(I)。

下面画出迭代后个体的目标函数值分布图和遗传算法性能跟踪图。遗传算法的运行结果如下：

（1）图 7.1 为目标函数 $f(x)=x\sin(10\pi\cdot x)+2.0$，$x\in[-1, 2]$的图像。

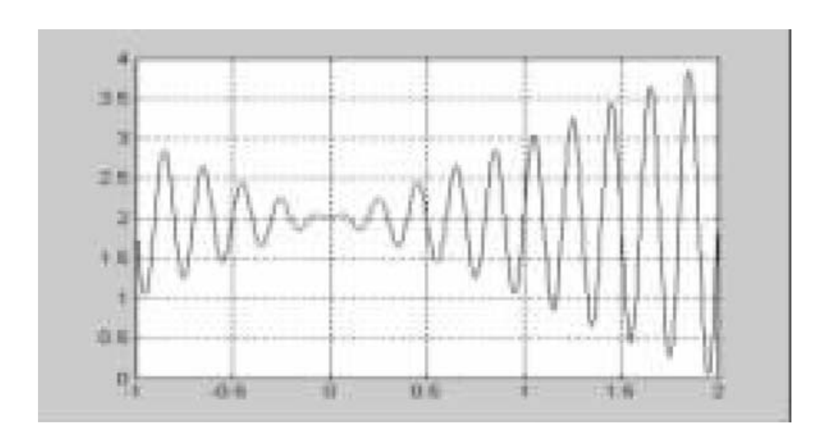


图 7.1　目标函数图像

（2）图 7.2 为目标函数的图像和初始随机种群个体分布图。

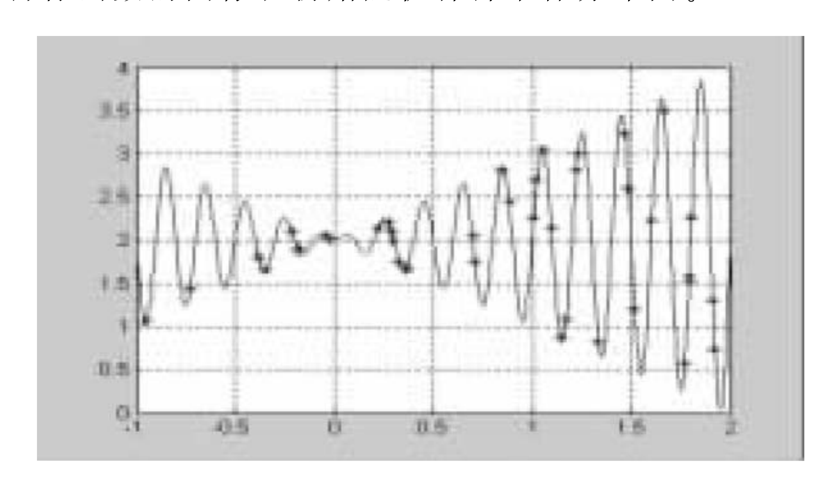


图 7.2　初始种群分布图

（3）经过一次遗传迭代后，寻优结果如图 7.3 所示。此时，$x=1.6357$，$f(x)=3.4729$。

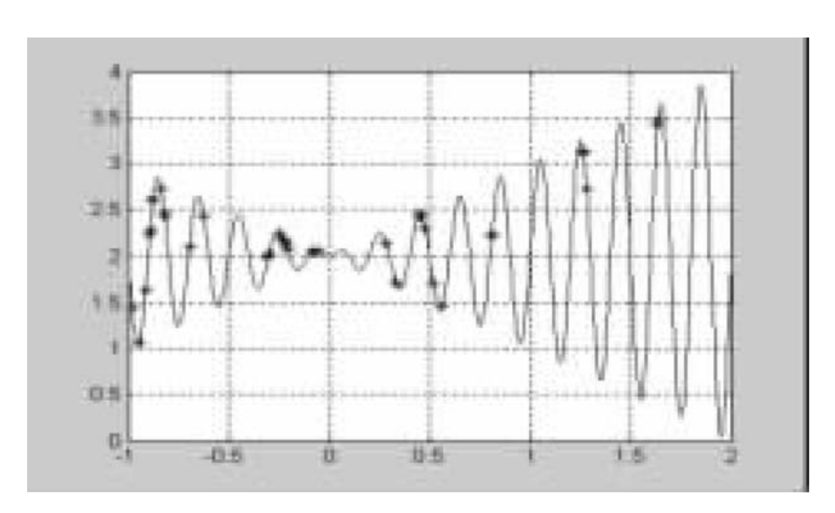


图 7.3　一次遗传迭代后的结果

（4）经过 10 次遗传迭代后，寻优结果如图 7.4 所示。此时，$x=1.8518$，$f(x)=3.8489$。

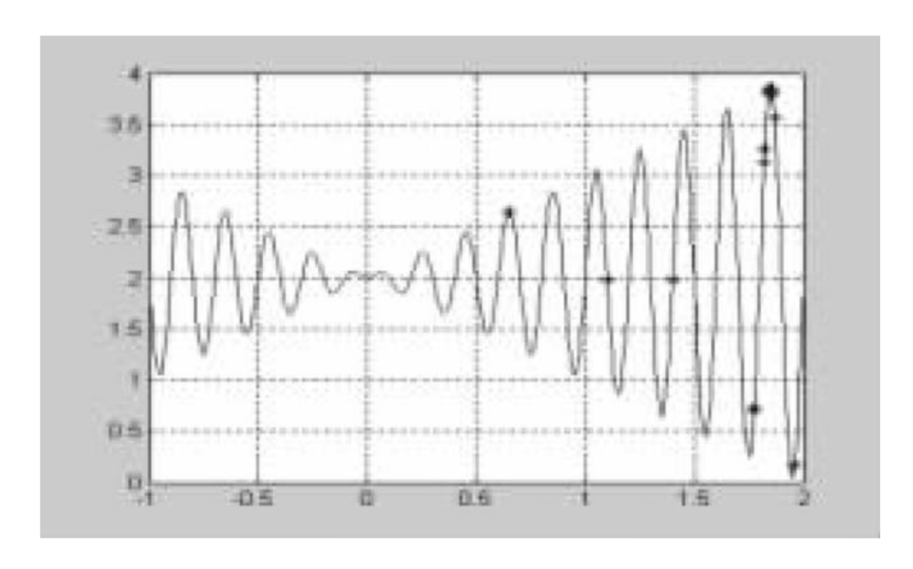


图 7.4　经过 10 次遗传迭代后的结果

（5）经过 25 次遗传迭代后，寻优结果如图 7.5 所示。此时，$x=1.8505$，$f(x)=3.8503$。

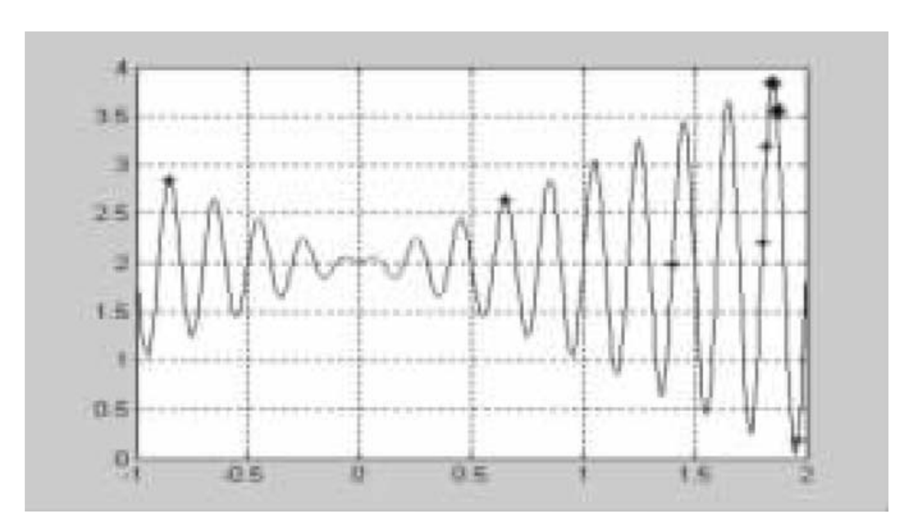


图 7.5　经过 25 次遗传迭代后的结果

（6）经过 25 次迭代后，最优解的变化和种群均值的变化见图 7.6。

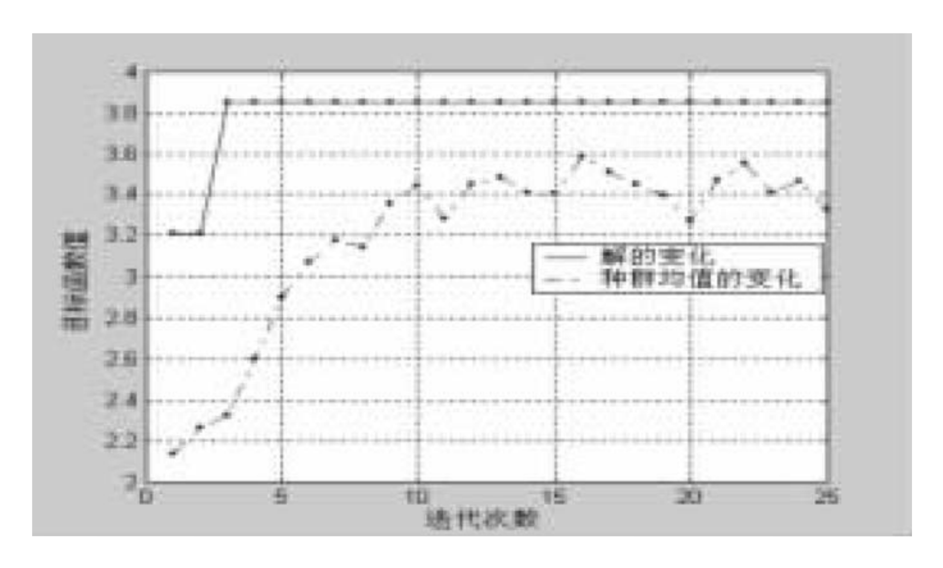



图 7.6　经过 25 次迭代后最优解的变化和种群均值的变化

## 7.2 多元单峰函数的优化实例

目标函数是 De Jong 函数，是一个连续、凸起的单峰函数，它的 M 文件 objfun1 包含在 GA 工具箱软件中。

De Jong 函数的表达式为

$$f(x) = \sum_{i=1}^{n} x_i^2, \quad -512 \leqslant x_i \leqslant 512$$

这里 $n$ 是定义问题维数的一个值，本例中选取 $n=20$，求解 $\min f(x)$。

由 De Jong 函数的表达式可以看出，De Jong 函数是一个简单的平方和函数，只有一个极小点(0，0，…，0)，理论最小值为 $f(0，0，…，0)=0$。

程序的主要变量：个体的数量 NIND 为 40，最大遗传代数为 MAXGEN=300，变量维数为 NVAR=20，每个变量使用 20 位表示，即 PRECI=20，使用代沟 GGAP=0.9。

下面为求解 De Jong 函数最小值的 MATLAB 代码。

```
% 定义遗传算法参数
NIND=40;              % 个体数目(Number of individuals)
MAXGEN=500;           % 最大遗传代数(Maximum number of generations)
NVAR=20;              % 变量的维数
PRECI=20;             % 变量的二进制位数(Precision of variables)
GGAP=0.9;             % 代沟(Generation gap)
trace=zeros(MAXGEN,2);
% 建立区域描述器(Build field descriptor)
FieldD=[rep ([PRECI],[1,NVAR]);rep ([-512;512],[1,NVAR]);rep ([1;0;1;1],
[1,NVAR])];
Chrom=crtbp (NIND, NVAR * PRECI);                     % 创建初始种群
gen=0;                                                % 代计数器
ObjV=objfun1(bs2rv (Chrom,FieldD));                   % 计算初始种群个体的目标函数值
while gen < MAXGEN,                                   % 迭代
    FitnV=ranking (ObjV);                             % 分配适应度值(Assign fitness values)
    SelCh=select ('sus', Chrom, FitnV, GGAP);         % 选择
    SelCh=recombin ('xovsp',SelCh,0.7);               % 重组
    SelCh=mut (SelCh);                                % 变异
    ObjVSel=objfun1 (bs2rv (SelCh,FieldD));           % 计算子代目标函数值
    [Chrom ObjV]=reins (Chrom,SelCh,1,1,ObjV,ObjVSel);       % 重插入
    gen=gen+1;                                               % 代计数器增加
    % 输出最优解及其对应的20个自变量的十进制值,Y为最优解,I为种群的序号
    trace (gen,1)=min (ObjV);                                % 遗传算法性能跟踪
    trace (gen,2)=sum (ObjV)/length (ObjV);
end
plot (trace (:,1));hold on;
plot (trace (:,2),'-.');grid;
legend ('种群均值的变化','解的变化').
```

区域描述器的构建：采用矩阵复制函数 rep 建立矩阵 FieldD，描述染色体的表示和解释。

一个初始种群被函数 crtbp 创建，随后产生一个矩阵 Chrom，它由 NIND 个均匀分布、长度为 NVAR * PRECI 的二进制串构成。

使用函数 objfun1 重新计算目标函数，初始种群中的所有个体的目标函数值由下面的程序段计算：

```
ObjV=objfun1(bs2rv (Chrom, FieldD));
```

函数 bs2rv 根据域描述器 FieldD 转换矩阵 Chrom 的二进制串为实值，返回一实值表现型的矩阵。这个 bs2rv 返回值矩阵通过直接作为目标函数 objfun1 的输入变量，将目标函数结果返回在矩阵 ObjV 中。

这个例子中基于排序的适应度分配计算由下面的程序段实现：

```
FitnV=ranking (ObjV);
```

对这个等级评定算法的缺省设置是选择等差 2，使用线性评估，给最适应个体的适应度值为 2，最差个体的适应度值为 0。适应度值结果被向量 FitnV 返回。

使用高级函数选择调用低级函数随机遍历抽样例程 sus，程序段为

```
SelCh=select('sus', Chrom, FitnV, GGAP);
```

后面的选择中，SelCh 包含来自原始染色体的 GGAP * NIND 个个体，这些个体将使用高级函数 recombin 进行重组，程序段为

```
SelCh=recombin ('xovsp', SelCh, 0.7);
```

函数 recombin 使个体通过 SelCh 被选择再生产，并使用单点交叉例程函数 xovsp，使用交叉概率 Px =0.7 执行交叉。个体作为矩阵 SelCh 的输入被排序，以便使奇数位置的个体与它相邻的个体进行交叉，如果 SelCh 个体的数量是奇数个，则最后一个个体不进行交叉而返回。交叉后产生的子代被同一个矩阵 SelCh 返回，实际使用的交叉例程通过支持使用不同函数名字串传递给 recombin 而改变。

为了产生一组子代，现在使用变异函数 mut 进行变异：

```
SelCh=mut (SelCh);
```

子代再次由矩阵 SelCh 返回，函数调用中变异概率没有被指定，这个缺省值 PM=0.7/Lind=0.0017，这里 Lind 是假定的个体长度。

子代的目标函数值 ObjVSel 由程序段计算：

```
ObjVSel=objfun1 (bs2rv (SelCh, FieldD));
```

因为使用了代沟，子代的数量比当前种群数量要小，所以使用恢复函数 reins 完成：

```
[Chrom,ObjV]=reins (Chrom, SelCh,1,1,ObjV,ObjVSel);
```

这里，Chrom 和 SelCh 是矩阵包含的原始种群和子代结果。这两个事件的第一个被使用单个种群和采用基于适应度的恢复，基于适应度的恢复用 SelCh 中的个体代替 Chrom 中最不适应的个体。原始种群目标函数值 ObjV 随后被要求作为 reins 的参数。另外，新种群的目标函数值不用重新计算整个种群的目标函数而返回，子代的目标值 ObjVSel 被提供。reins 返回具有插入子代的新种群 Chrom 和这个种群的目标函数值 ObjV。

这个遗传优化的结果包含在矩阵 ObjV 中，决策变量的值可能被包含在下面的程序段中：

Phen＝bs2rv（Chrom，FieldD）；

遗传算法的运行结果如下：

（1）图 7.7 为二维 De Jong 函数图形。

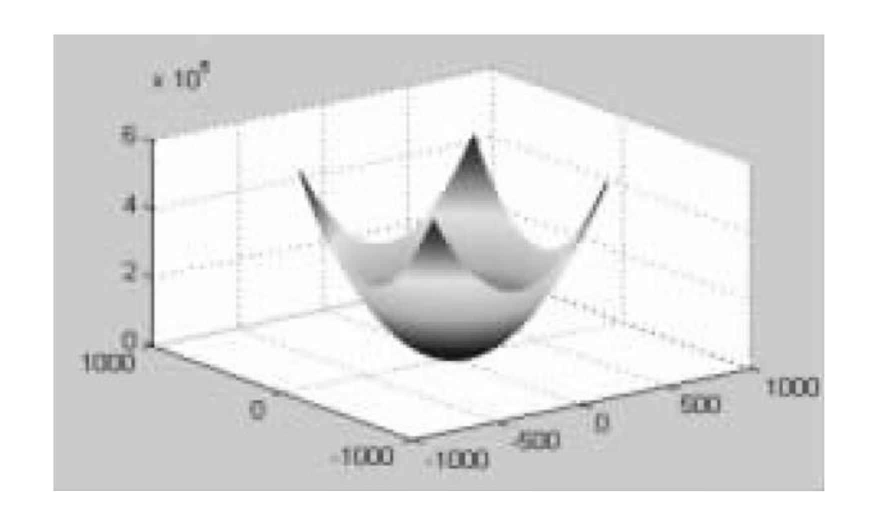


图 7.7 二维 De Jong 函数图形

（2）初始种群中个体的目标函数值分布图如图 7.8 所示。

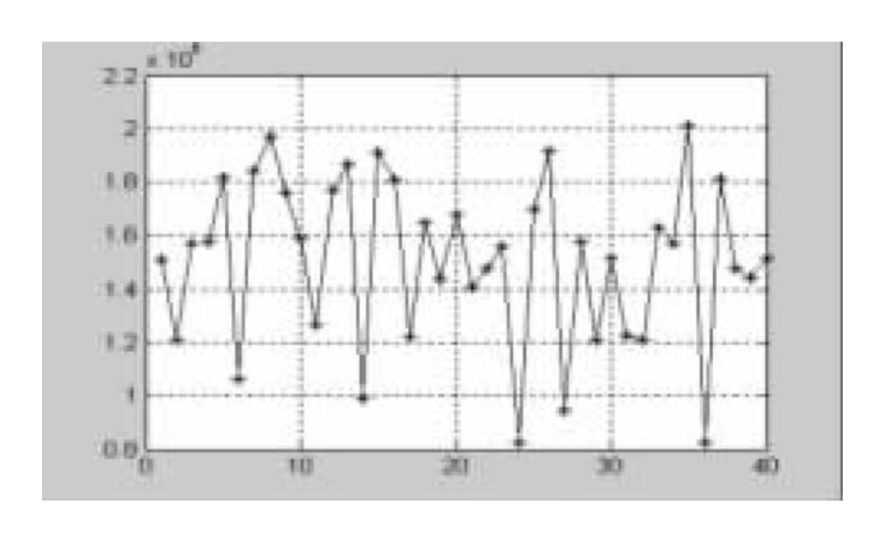


图 7.8 初始种群中个体的目标函数值分布图

（3）经过 20 次遗传迭代后的结果如图 7.9 所示。此时，$\min f(x)=2.7412e+5$。

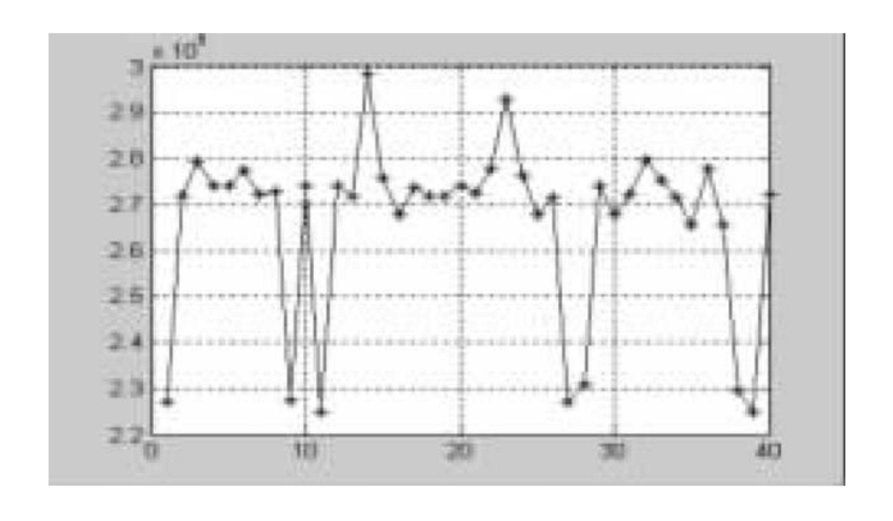


图 7.9 经过 20 次遗传迭代后的结果

（4）经过 100 次遗传迭代后的结果如图 7.10 所示。此时，$\min f(x)=6.9666\mathrm{e}+3$。

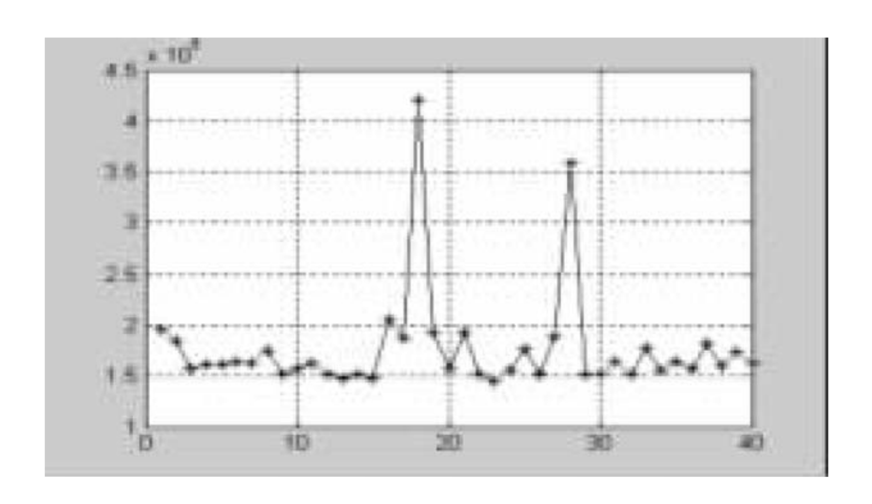


图 7.10　经过 100 次遗传迭代后的结果

（5）经过 250 次遗传迭代后的结果如图 7.11 所示。此时，$\min f(x)=409.4516$。

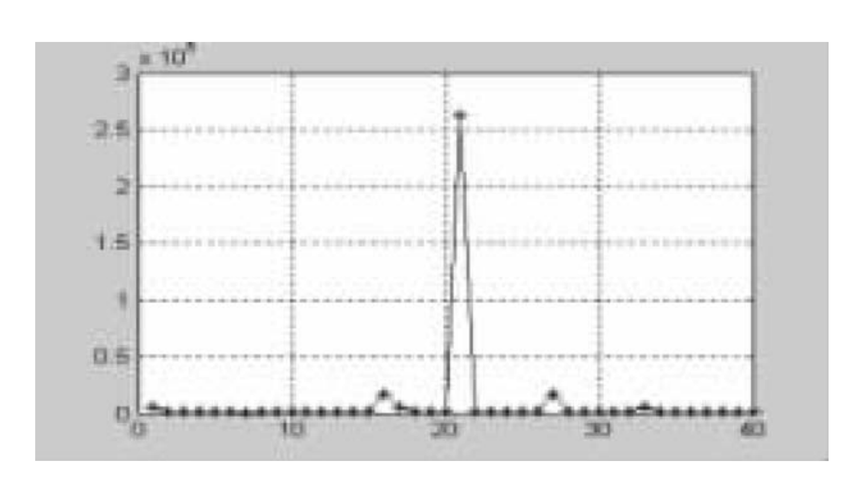


图 7.11　经过 250 次遗传迭代后的结果

（6）经过 500 次遗传迭代后的结果如图 7.12 所示。此时，$\min f(x)=4.2550$。

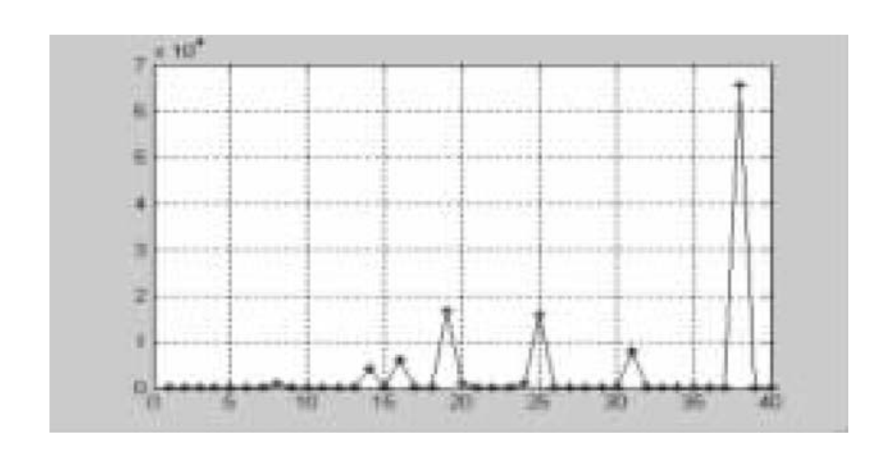


图 7.12　经过 500 次遗传迭代后的结果

（7）经过 250 次迭代后种群目标函数均值的变化和最优解的变化如图 7.13 所示。$\min f(x)=4.2550$ 时 20 个变量的值见表 7.1。

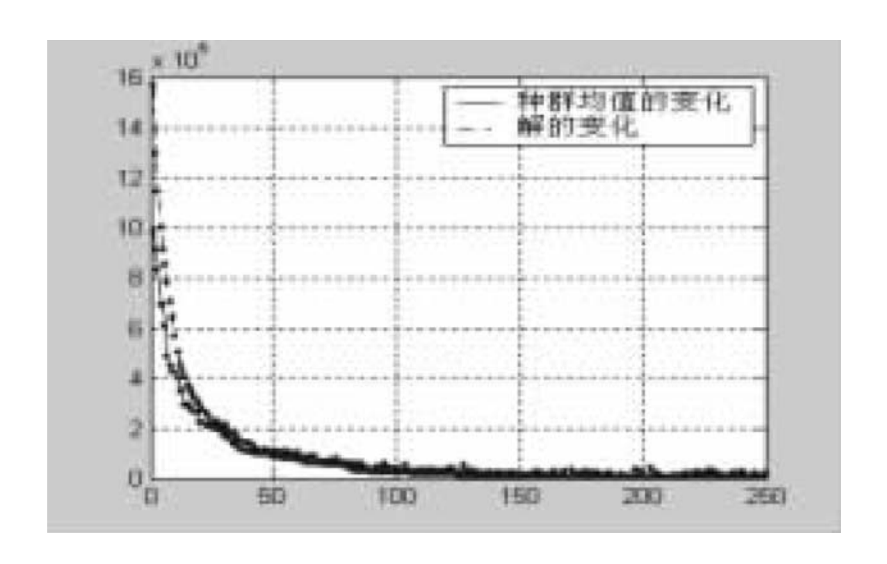



图 7.13　经过 250 次迭代后种群目标函数均值的变化和最优解的变化

表 **7.1**　目标函数取最优解时 **20** 个变量的值

| 序号 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 变量值 | −2.0015 | 0.1362 | −0.0190 | −0.0171 | −0.0659 | −0.1382 | −0.0181 | −0.0610 | −0.0112 | 0.1274 |
| 序号 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 变量值 | −0.0142 | 0.0640 | 0.0474 | 0.0122 | −0.0962 | 0.2524 | −0.0708 | 0.2690 | −0.1411 | −0.0952 |

## 7.3　多元多峰函数的优化实例

Shubert 函数为

$$f(x_1, x_2) = \sum_{i=1}^{5} i\cos[(i+1)\cdot x_1 + i] \cdot \sum_{i=1}^{5} i\cos[(i+1)\cdot x_2 + i]$$

$$-10 \leqslant x_1, x_2 \leqslant 10$$

求 $\min f(x)$。

图 7.14 所示为 Shubert 函数图像。

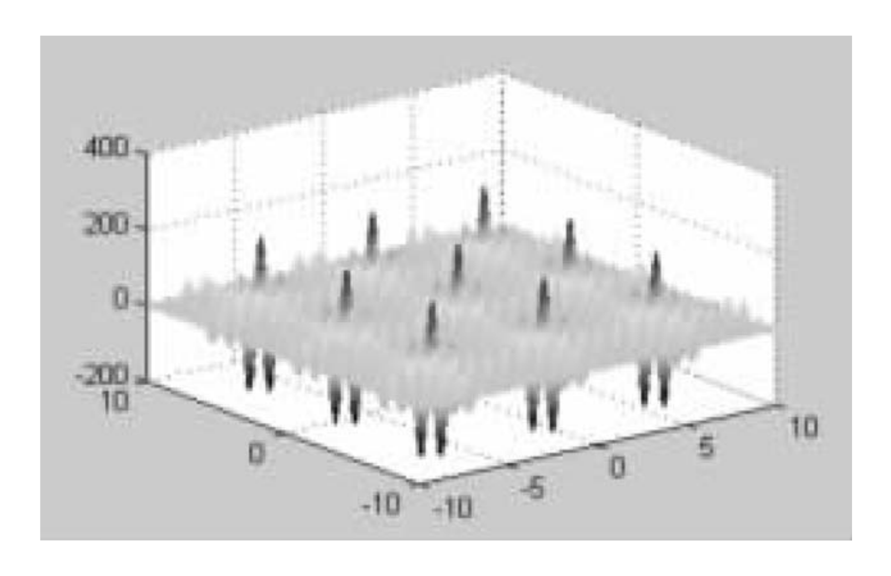


图 7.14　Shubert 函数图像

下面为 Shubert 函数寻优的 MATLAB 代码。

```
function z=Shubert (x,y)                    % Shubert 函数
z=((1 * cos ((1+1) * x+1))+ (2 * cos ((2+1) * x+2))+ (3 * cos ((3+1) * x+3))+...
   (4 * cos((4+1) * x+4))+ (5 * cos ((5+1) * x+5))) * ((1 * cos ((1+1) * y+1))+...
   (2 * cos ((2+1) * y+2))+ (3 * cos ((3+1) * y+3))+ (4 * cos ((4+1) * y+4))+ ...
   (5 * cos((5+1) * y+5)));
[x1,x2]=meshgrid (-10:.1:10);
Figure(1);mesh (x1,x2,Shubert (x1,x2));       % 画出 Shubert 函数图像
% 定义遗传算法参数
NIND=40;          % 个体数目(Number of individuals)
MAXGEN =50;  % 最大遗传代数(Maximum number of generations)
NVAR=2;           % 变量数目
PRECI=25;         % 变量的二进制位数(Precision of variables)
GGAP=0.9;         % 代沟(Generation gap)
% 建立区域描述器(Build field descriptor)
FieldD=[rep ([PRECI],[1,NVAR]);rep ([-3;3],[1,NVAR]);rep ([1;0;1;1],
            [1,NVAR])];
Chrom=crtbp(NIND, NVAR * PRECI);                 % 创建初始种群
gen=0;
trace=zeros (MAXGEN,2);                          % 遗传算法性能跟踪初始值
x=bs2rv (Chrom,FieldD);                          % 初始种群十进制转换
ObjV =Shubert (x (:,1),x (:,2));                 % 计算初始种群的目标函数值
while gen<MAXGEN,
    FitnV=ranking (ObjV);                        % 分配适应度值(Assign fitness values)
    SelCh=select ('sus', Chrom, FitnV, GGAP);    % 选择
    SelCh=recombin ('xovsp',SelCh,0.7);          % 重组
    SelCh=mut (SelCh);                           % 变异
    x=bs2rv (SelCh,FieldD);                      % 子代十进制转换
    ObjVSel =Shubert (x (:,1),x (:,2)) ;         % 重插入
    [Chrom ObjV]=reins (Chrom,SelCh,1,1,ObjV,ObjVSel);
    gen=gen+1;
    [Y,I]=min (ObjVSel);
    Y, bs2rv (Chrom (I,:),FieldD)) ;             % 输出最优解及其对应的自变量值
    trace (gen,1)=min (ObjV);
    trace (gen,2)=sum (ObjV)/length (ObjV);      % 遗传算法性能跟踪
    if (gen==50)                                 % 迭代数为 50 时画出目标函数值分布图
        figure (2);
        plot (ObjV);hold on;
        plot (ObjV,'b * ');grid;
    end
end
figure (3);clf;
plot (trace (:,1));hold on;
plot (trace (:,2),'-.');grid;
legend ('解的变化','种群均值的变化').
```

遗传算法的显示结果如下：

(1) 初始种群的目标函数值分布如图 7.15 所示。

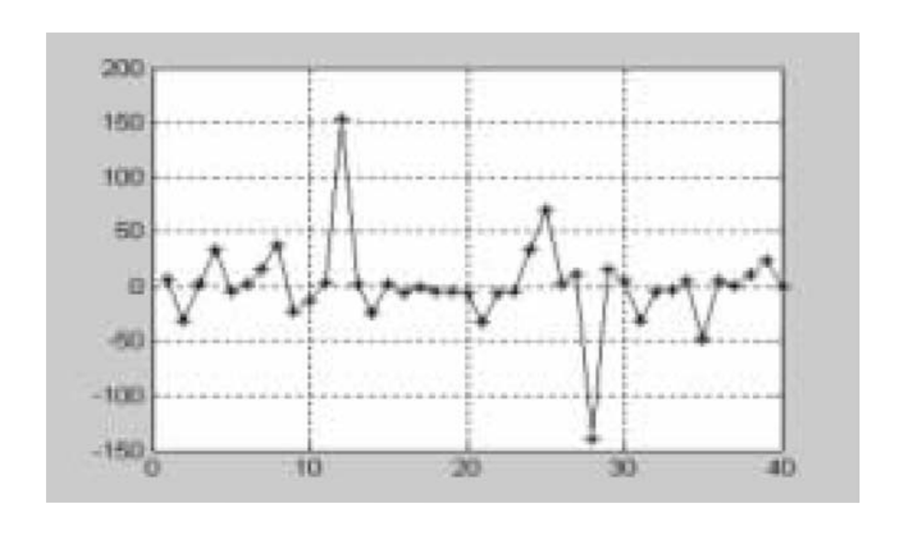



图 7.15 初始种群的目标函数值

(2) 经过一次迭代后的目标函数值如图 7.16 所示。此时，$x_1=-1.3900$，$x_2=-2.9601$，$\min f(x)=-44.5224$。

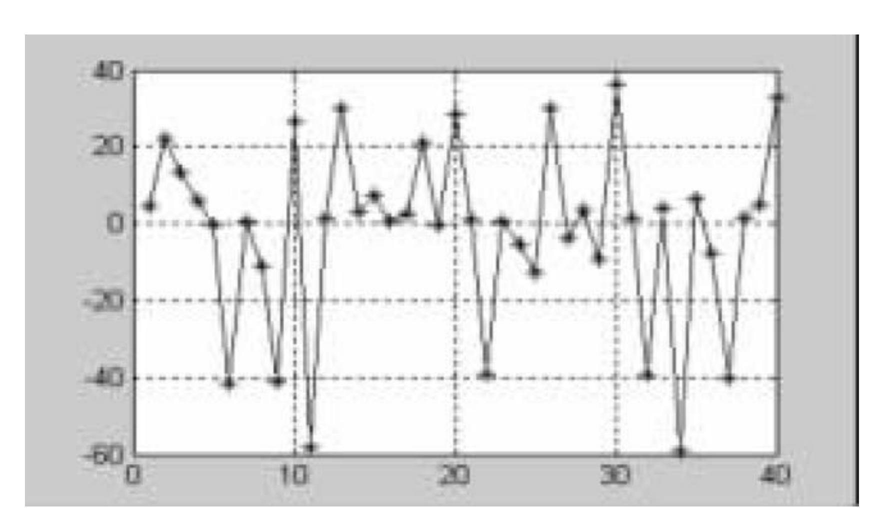



图 7.16 经过一次迭代后的结果

(3) 经过 10 次迭代后的目标函数值如图 7.17 所示。此时，$x_1=-1.3811$，$x_2=0.8980$，$\min f(x)=-186.0739$。

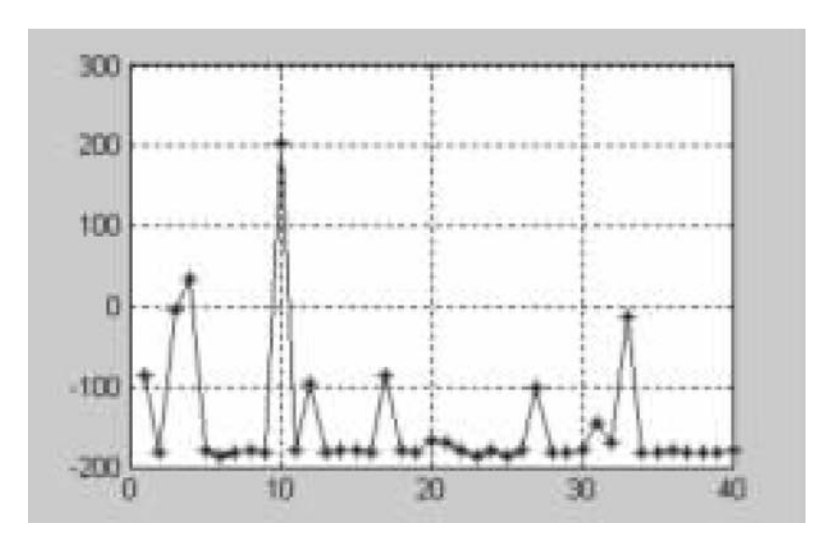



图 7.17 经过 10 次迭代后的结果

(4) 经过 50 次迭代后的目标函数值如图 7.18 所示。此时，$x_1 = -1.4342$，$x_2 = -0.8003$，$\min f(x) = -186.7309$。

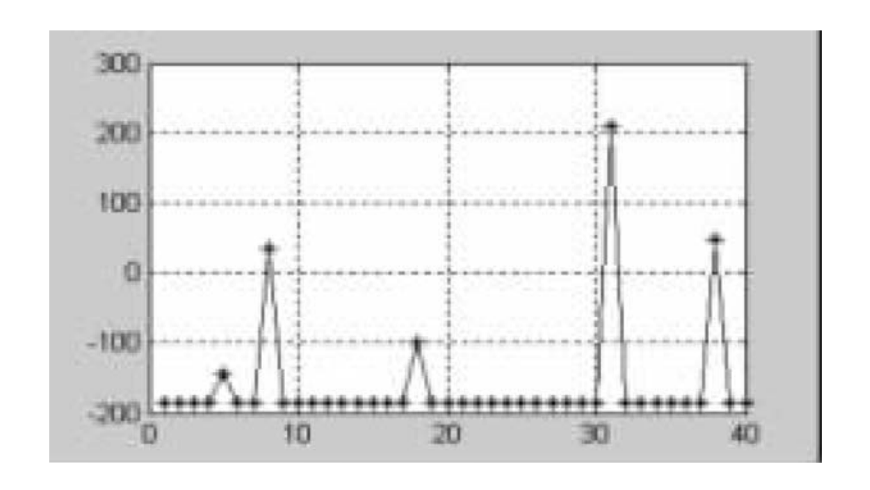



图 7.18 经过 50 次迭代后的结果

(5) 经过 50 次迭代后，种群目标函数均值的变化和最优解的变化如图 7.19 所示。

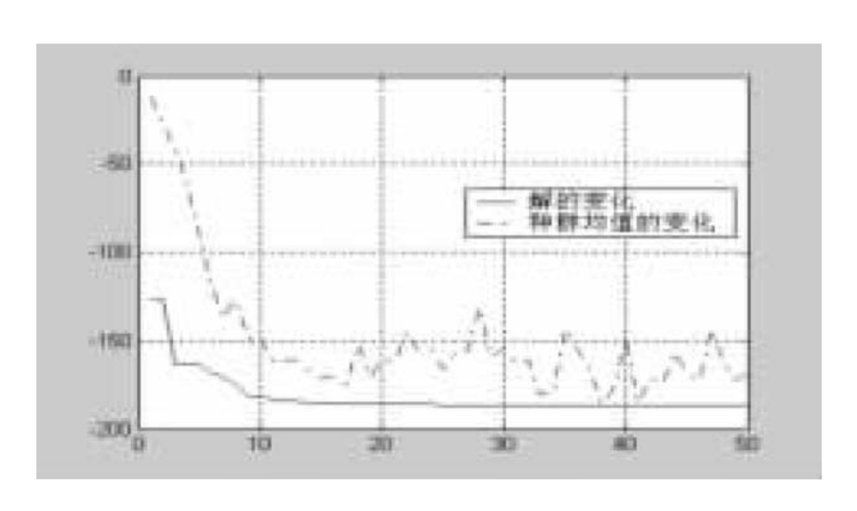



图 7.19 经过 50 次迭代后种群目标函数均值的变化和最优解的变化

## 7.4 收获系统最优控制

收获系统(Harvest)是一个一阶的离散方程，表达式为

$$\begin{cases} x(k+1) = a * x(k) - u(k) \\ \text{s.t.} \qquad x(0) = x(N) \end{cases} \quad k = 1, 2, \cdots, N$$

这里，$x(0)$ 是一个初始的状态条件，$a$ 是一个刻度常量，$x(k) \in \mathbf{R}$ 和 $u(k) \in \mathbf{R}^+$ 分别是状态和非常控制，$N$ 是解决问题使用的步骤数。

目标函数为

$$f(u) = \sum_{k=1}^{N} \sqrt{u(k)}$$

这个问题的精确优化解答可由下式确定：

$$\max f(x) = \sqrt{\frac{x(0)(a^N - 1)^2}{a^{N-1}(a-1)}}$$

GA 工具箱提供了一个 M 文件 objharv，作为目标函数。注意，这里是一个最大化问

题，工具箱例程实现是最小化问题，这个目标函数 objharv，即 $f(u)$乘以$-1$后可转化为最小化问题。初始条件 $x(0)$设为 100，$a$ 取为 1.1。

这个问题的控制步骤 $N=20$，因此使用的决策变量个数 NVAR＝20，作为每个控制输入 $u(k)$。决策变量被限制在 RANGE＝[0，200]范围内，限制最大的控制输入在任意步时为 200，这个区域描述器 FieldD 描述的决策变量可以使用矩阵复制函数 rep 构造。

这个 GA 的参数可用 MATLAB 变量来指定。

除了传统 GA 参数，例如代沟（GGAP）和交叉概率（XOVR)外，还有大量与多种群 GAS 有关的其他参数被定义。这里 INSR＝0.9，说明每一代中只有 90％的个体被复制到种群中，SUBPOP＝8，说明 8 个子种群使用的迁移概率为 MIGR ＝0.2 或 20％，每 20 代(MIGGEN＝ 20)在子种群与当前迁移之间。每个子种群包含 20 个个体，NIND＝20。

脚本文件使用的函数使用 MATLAB 串指定。用串 SEL_F 、XOV_F、MUT_F、OBJ_F 指定分别代表选择函数名、重组函数名、变异函数名、目标函数名。

由于使用离散重组函数 recdis 进行子代的培育，交叉概率没有使用，XOVR＝1。

初始种群的创建使用函数 crtrp，代计数器 gen 设置为 0。

在由 FieldD 指定的范围内使用一致性随机挑选个体决策变量组成 SUBPOP * NIND 个个体。矩阵 Chrom 包含了所有子种群并且子种群中所有个体的目标函数值能使用 MATLAB 的 feval 命令直接计算。ObjV＝feval (OBJ_F，Chrom)中 feval 执行函数计算获得第一个输入变量，这里是目标函数名 objharv，包含在 OBJ_F 中，这个函数将被计算并用剩余的参数作为输入调用这个函数。这里函数调用是：

ObjV＝objharv (Chrom)；

由于使用实值编码，不需要将染色体转换为表现型表示。GA 现在将进入代循环。

下面为收获系统最优控制的 MATLAB 代码。

```
% 定义遗传算法参数
NVAR=20;                % 变量维数
RANGE=[0;200];          % 变量范围
GGAP=0.8;               % 代沟(Generation gap)
XOVR=1;                 % 交叉率
MUTR=1/NVAR;            % 变异率
MAXGEN=500;             % 最大遗传代数(Maximum number of generations)
INSR=0.9;               % 插入率
SUBPOP =8;              % 子种群数
MIGR=0.2;               % 迁移率
MIGGEN=20;              % 在子种群与迁移之间 20 代
NIND=20;                % 个体数目(Number of individuals)
SEL_F='sus';            % 选择函数名
XOV_F ='recdis';        % 重组函数名
MUT_F ='mutbga';        % 变异函数名
OBJ_F ='objharv';       % 目标函数名
FieldDD=rep (RANGE,[1,NVAR]);            % 译码矩阵
gen=0;
trace=zeros (MAXGEN,2);                  % 遗传算法性能跟踪
```

```
Chrom=crtrp (NIND,FieldDD);                      % 创建初始种群
ObjV=objharv (Chrom);                            % 计算目标函数值
while gen < MAXGEN,                              % 代循环
    FitnV=ranking (ObjV,[2,1],SUBPOP);  % 分配适应度值(Assign fitness values)
    SelCh=select (SEL_F, Chrom, FitnV, GGAP, SUBPOP);        % 选择
    SelCh=recombin (XOV_F, SelCh, XOVR, SUBPOP);             % 重组
    SelCh=mutate (MUT_F,SelCh,FieldDD,[MUTR],SUBPOP);  % 变异
    ObjVOff=feval (OBJ_F,SelCh);                             % 计算目标函数值
    [Chrom, ObjV]=reins (Chrom, SelCh, SUBPOP,[1 INSR], ObjV, ObjVOff); % 替代
    gen=gen+1;
    [trace (gen,1),I]=min (ObjV);
    trace (gen,2)=mean (ObjV);
    % 在子种群之间迁移个体
        if (rem (gen,MIGGEN) == 0)
            [Chrom, ObjV]=migrate (Chrom, SUBPOP, [MIGR, 1, 1], ObjV);
        end
    end
[Y,I]=min (ObjV);              % 输出最优解及其序号, Y 为最优解, I 为种群的序号
figure (1);plot (Chrom (I,:));
hold on;grid;
plot (Chrom (I,:),'bo')
figure (2);plot (-trace (:,1));
hold on;
plot (-trace (:,2),'-.');                % 遗传算法性能跟踪分布图
legend ('解的变化','种群均值的变化');xlabel ('迭代次数')
```

由于使用多子种群，待评定要求指定必须的选择压位差，这里选择压位差为 2。指定子种群的个数为 SUBPOP。每个子种群的个体的目标值 ObjV 将独立排序，适应度值结果集将由向量 FitnV 返回。

在每个子种群中，个体将由高级选择函数 select 选择独立地培育子代：

SelCh=select (SEL_F, Chrom, FitnV, GGAP, SUBPOP);

Select 为每个子种群调用低级选择函数 SEL_F ='sus'，并建立包含用于重组的所有个体对的矩阵 SelCh，代沟 GGAP=0.8，意思就是 0.8×20=16，GGAP * NIND 个个体从每个子种群中选择，SelCh 包含 GGAP * NIND * SUBPOP=128 个个体。

高级重组函数 recombin 用于重组 SelCh 中每个子种群中的每对个体。

SelCh=recombin (XOV_F, SelCh, XOVR, SUBPOP);

重组函数 XOV_F='recdis'对子种群中个体对执行离散重组。由于离散重组不需要指定常规的交叉概率，这里使用变量 XOVR=1.0 仅仅是为了兼容。

子代现在将变异：

SelCh=mutate (MUT_F,SelCh,FieldD,MUTR,SUBPOP);

这里 GA 育种器变异函数 MUT_F ='mutbga'使用高级变异例程 mutate 调用，使用变异概率 MUTR=1/NIND=0.05。这个 GA 育种器变异函数需要域描述器 FieldD，以使

变异还会产生超出决策变量边界的结果。

所有子代的目标值 ObjVOff 现在可以再用 feval 计算：

```
ObjVOff=feval (OBJ_F, SelCh);
```

子代现在可以插入适当的子种群中：

```
[Chrom,ObjV]=reins (Chrom, SelCh,SUBPOP,[1,INSR],ObjV,ObjVOff);
```

使用基于适应度的插入，但对 reins 的第四个自变量的附加扩展参数指定了插入概率 INSR＝0.9。这里的意思是指每个子种群的最小适应度的10％子代不会被插入。

多种群 GA 的个体在种群之间以相同间隔迁移。工具箱的迁移例程用于在子种群间根据相同的迁移策略交换个体，在这个例子中，每20代(MIGGEN＝20)在子种群间发生迁移。子种群间的迁移语句为

```
if (rem (gen, MIGGEN) == 0)
    [Chrom, ObjV]=migrate (Chrom, SUBPOP, [MIGR, 1, 1], ObjV);
end
```

在这里子种群中最适应的20％(MIGR＝0.2)的个体被选择迁移。最邻近的子种群在它们之间交换个体，均匀地重插入移民个体。返回矩阵 Chrom 和向量 ObjV 作为迁移结果反映子种群中个体的变化。

GA 迭代循环直到 gen＝MAXGEN 并随后终止。

GA 搜索获得的结果被包含在矩阵 ObjV 中。最佳个体的目标值和索引号可用函数 min 搜索。例如：

```
[Y, I]=min (ObjV);
```

运行后得到：Y＝－73.2370，I＝50。

注意：改变目标函数的符号形成最小化问题，这个结果相当于目标函数值为73.2370，给出的精确解答应为73.2376。因此这个 GA 优化解与精确解之间的误差在$10^{-5}$以内。染色体值显示使用如下命令：

```
plot (Chrom (I,:)).
```

遗传算法的显示结果如下：

(1) 图7.20为初始随机种群个体分布图。

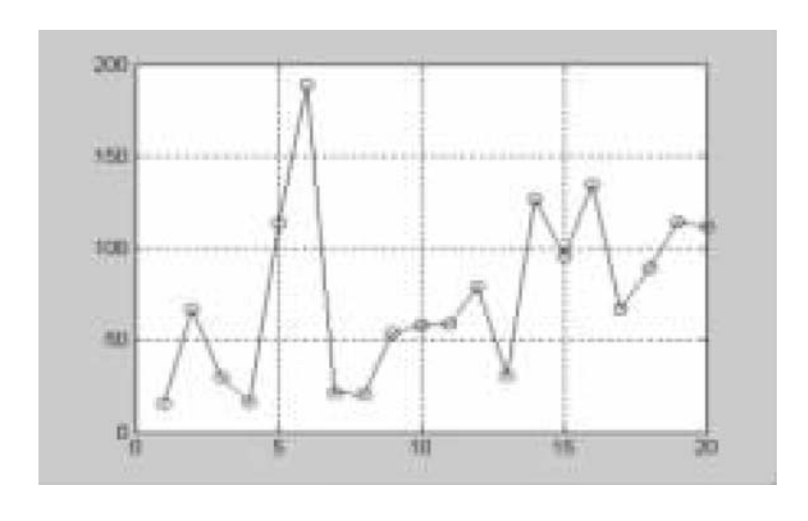



图7.20　初始种群的分布图

(2) 经过50次迭代后，寻优结果如图7.21所示。此时，$\max f(x)=9.1760$。

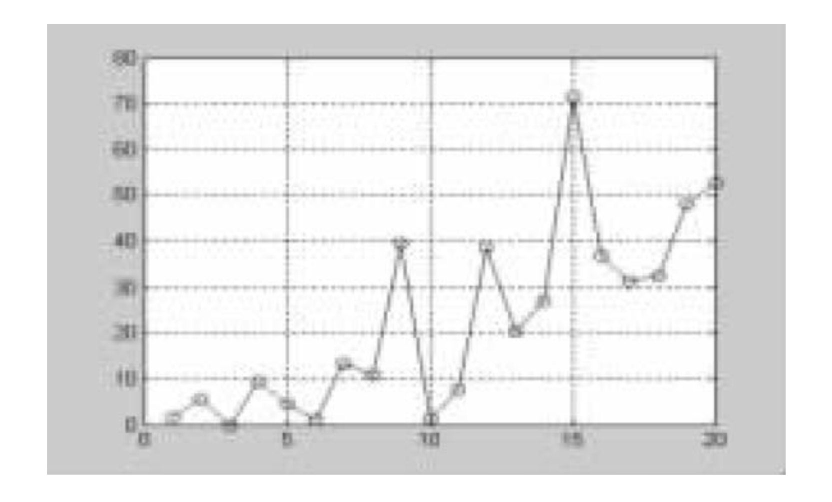



图 7.21　经过 50 次迭代后的优化解

（3）经过 200 次迭代后，寻优结果如图 7.22 所示。此时，$\max f(x)=58.8087$。

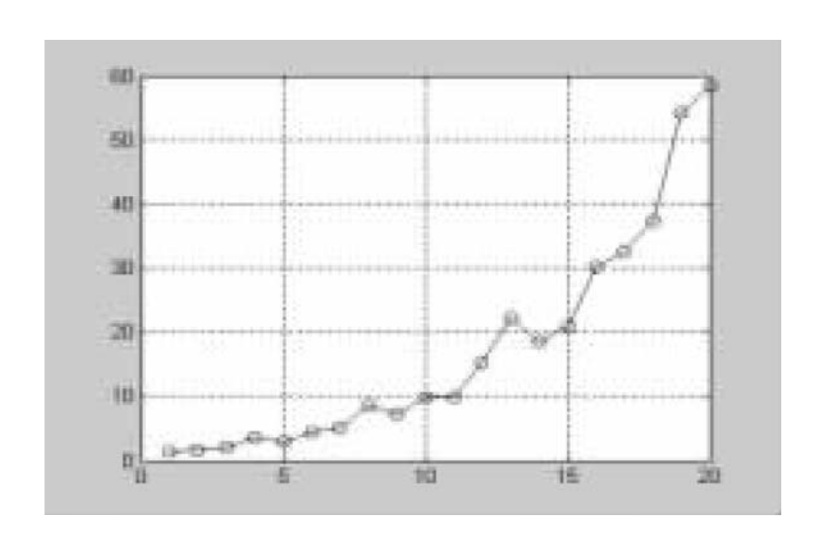



图 7.22　经过 200 次迭代后的优化解

（4）经过 1000 次迭代后，寻优结果如图 7.23 所示。此时，$\max f(x)=68.0320$。

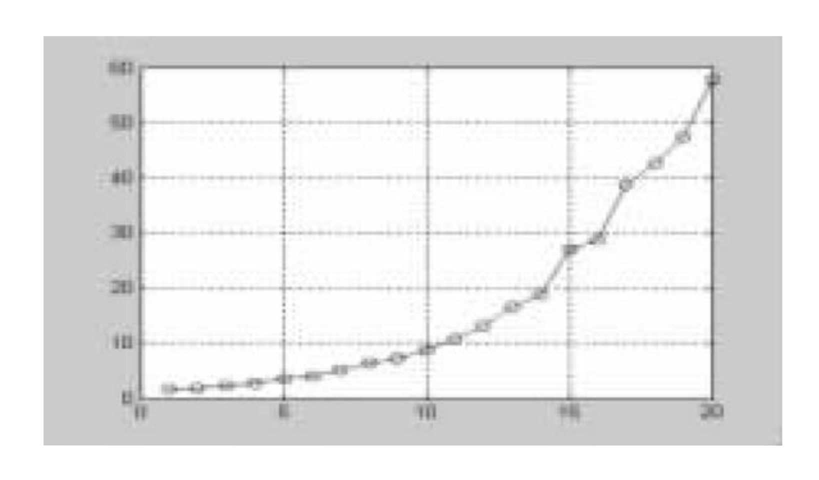



图 7.23　经过 1000 次迭代后的优化解

## 7.5　装载系统的最优问题

装载系统是一个二维系统，表达式如下：

$$\begin{cases} x_1(k+1) = x_2(k) \\ x_2(k+1) = 2 * x_2(k) - x_1(k) + \dfrac{1}{N^2}u(k) \end{cases} \quad k = 1, 2, \cdots, N$$

目标函数为

$$f(x, u) = -x_1(N+1) + \frac{1}{2N}\sum_{k=1}^{N} u^2(k)$$

理论最优解为

$$\min f(x, u) = -\frac{1}{3} + \frac{3N-1}{6N^2} + \frac{1}{2N^3}\sum_{k=1}^{N-1} k^2$$

一个实现本目标函数的 M 文件 objpush 包含在 GA 工具箱软件中。

下面为装载系统的最优问题的 MATLAB 代码。

```
%定义遗传算法参数
GGAP=0.8;              % 代沟(Generation gap)
XOVR=1;                % 交叉率
NVAR=20;               % 变量维数
MUTR=1/NVAR;           % 变异率
MAXGEN=200;            % 最大遗传代数(Maximum number of generations)
INSR=0.9;              % 插入率
SUBPOP =12;            % 子种群数
MIGR=0.2;              % 迁移率
MIGGEN=20;             % 每 20 代迁移个体
NIND =20;              % 个体数目(Number of individuals)
RANGE=[0;10];          % 变量范围
SEL_F='sus';           % 选择函数名
XOV_F ='recdis';       % 重组函数名
MUT_F ='mutbga';       % 变异函数名
OBJ_F ='objpush';      % 目标函数名
FieldDD=rep (RANGE,[1,NVAR]);
trace=zeros (MAXGEN,2);                        % 遗传算法性能跟踪
Chrom=crtrp (SUBPOP * NIND,FieldDD);           % 创建初始种群
gen=0;
ObjV=feval (OBJ_F,Chrom);
while gen < MAXGEN,                            % 代循环
    FitnV=ranking (ObjV,[2,0],SUBPOP);         % 分配适应度值(Assign fitness values)
    SelCh=select (SEL_F,Chrom,FitnV, GGAP,SUBPOP);          % 选择
    SelCh=recombin (XOV_F, SelCh,XOVR,SUBPOP);              % 重组
    SelCh=mutate (MUT_F,SelCh,FieldDD,[MUTR],SUBPOP);       % 变异
    ObjVOff=feval (OBJ_F,SelCh);                            % 计算子代目标函数值
    [Chrom, ObjV]=reins (Chrom, SelCh, SUBPOP,[1 INSR], ObjV, ObjVOff);% 替代
    gen=gen+1;
    [trace (gen,1),I]=min (ObjV);
    trace (gen,2)=mean (ObjV);
```

```
end
[Y,I]=min (ObjV);                              % 最优控制向量值及其序号
subplot (211);                                 % 最优控制向量分布图
plot (Chrom (I,:));hold on;
plot (Chrom (I,:),'.');grid
subplot (212);                                 % 遗传算法性能跟踪分布图
plot (trace (:,1));hold on;
plot (trace (:,2),'-');grid
legend ('解的变化','种群均值的变化')
```

遗传算法的显示结果如下：

(1) 经过 50 次迭代后优化解的目标函数值及性能跟踪如图 7.24 所示。此时，$\min f(x, u)=0.0369$。

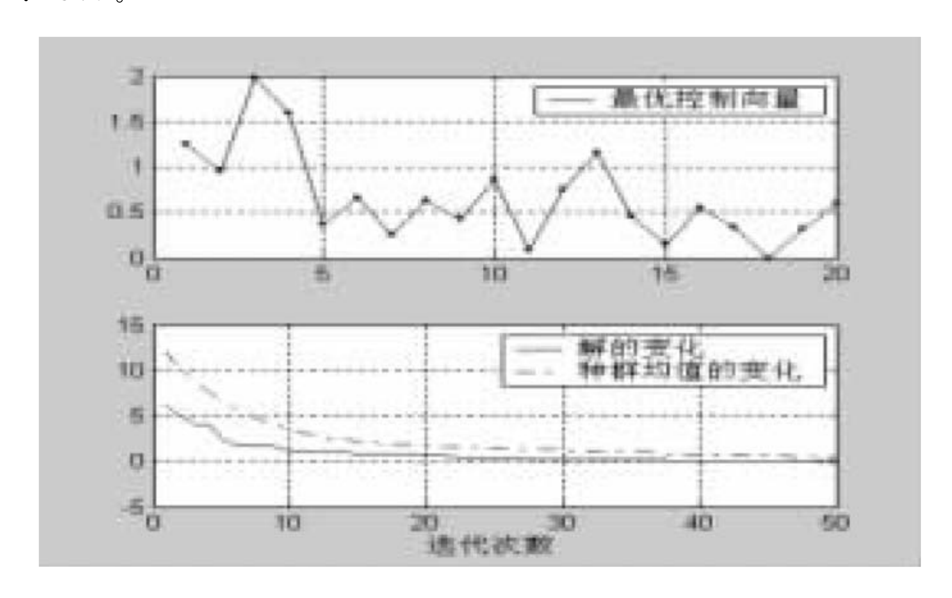



图 7.24　经过 50 次迭代后的优化解的目标函数值及性能跟踪

(2) 经过 100 次迭代后优化解的目标函数值及性能跟踪如图 7.25 所示。此时，$\min f(x, u)=-0.1425$。

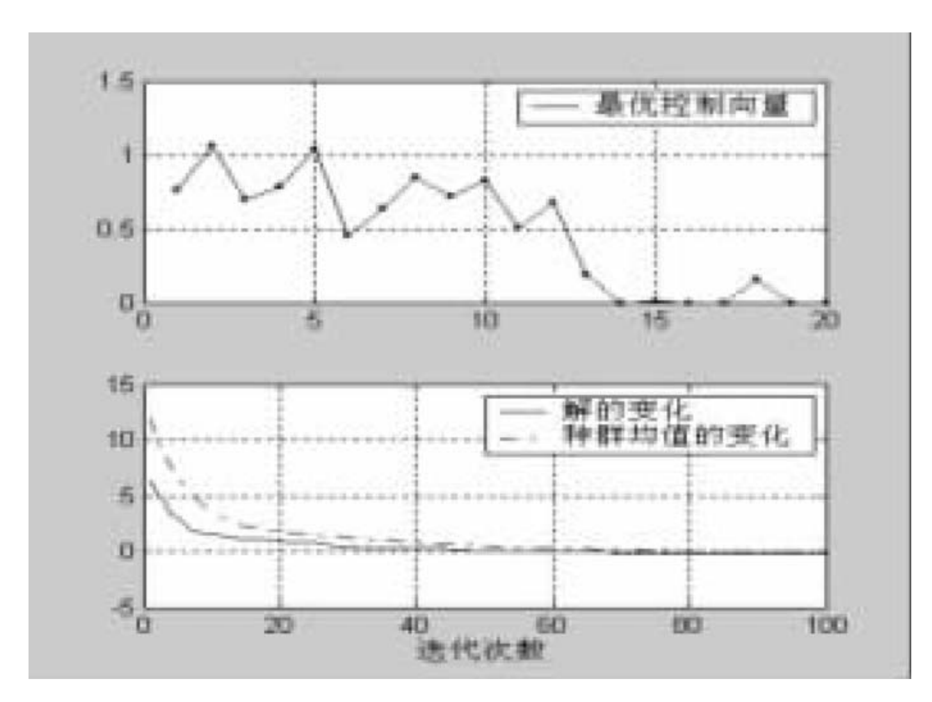



图 7.25　经过 100 次迭代后的优化解的目标函数值及性能跟踪

(3) 经过 200 次迭代后的优化解的目标函数值及性能跟踪如图 7.26 所示。此时，$\min f(x, u)=-0.1536$。

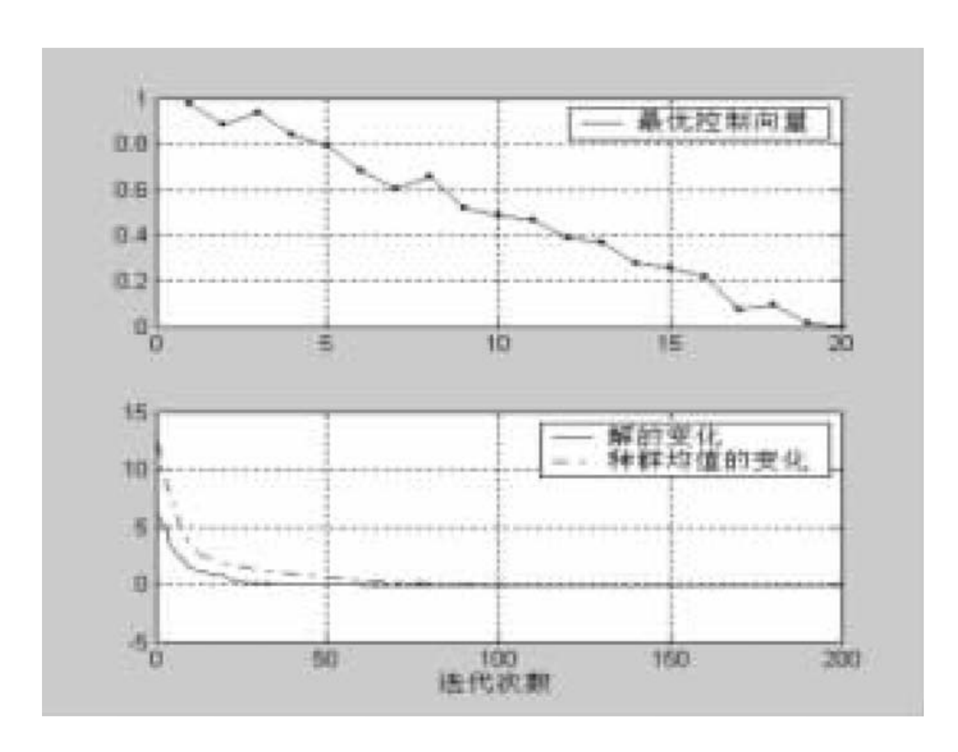



图 7.26 经过 200 次迭代后的优化解的目标函数值及性能跟踪

为了比较，计算出理论最优解：

$$\min f(x, u) = -\frac{1}{3} + \frac{3 \times 20 - 1}{6 \times 20^2} + \frac{1}{2 \times 20^3} \sum_{k=1}^{20-1} k^2 = -0.1544$$

经过 200 次迭代后的优化解 $-0.1536$ 与理论最优解 $-0.1544$ 之间的误差仅为 0.0008。

## 7.6 离散二次线性系统最优控制问题

假设二阶线性系统是一维的，其表达式为

$$x(k+1) = a * x(k) + b * u(k), \quad k = 1, 2, \cdots, N$$

目标函数定义为

$$f(x, u) = q * x(n+1)^2 + \sum_{k=1}^{N} [s * x(k)^2 + r * u(k)^2]$$

求 $\min f(x, u)$。

参数设置如表 7.2 所示。

表 **7.2** 参数设置

| 参数名 | $N$ | $x(0)$ | $s$ | $r$ | $q$ | $a$ | $b$ |
|---|---|---|---|---|---|---|---|
| 参数值 | 45 | 100 | 1 | 1 | 1 | 1 | 1 |

一个实现本目标函数的 M 文件 objlinq 包含在 GA 工具箱软件中。

下面为离散二次线性系统最优控制的 MATLAB 代码。

```
% 定义遗传算法参数
GGAP=0.8;          % 代沟(Generation gap)
XOVR=1;            % 交叉率
```

```
NVAR=45;              % 变量维数
MUTR=1/NVAR;          % 变异率
MAXGEN=2000;          % 最大遗传代数(Maximum number of generations)
INSR=0.9;             % 插入率
SUBPOP =12;           % 子代数目
MIGR=0.2;             % 迁移率
MIGGEN=20;            % 每 20 代迁移个体
NIND =20;             % 个体数目(Number of individuals)
SEL_F='sus';          % 选择函数名
XOV_F ='recdis';      % 重组函数名
MUT_F ='mutbga';      % 变异函数名
OBJ_F ='objlinq';     % 目标函数名
FieldDR=feval (OBJ_F,[ ],1);
Chrom=crtrp (SUBPOP * NIND,FieldDR);        % 创建初始种群
gen=0;
trace=zeros (MAXGEN,2);                     % 遗传算法性能跟踪
ObjV=feval (OBJ_F,Chrom);                   % 计算目标函数值
while gen < MAXGEN,                         % 代循环
    trace (gen+1,1)=min (ObjV);
    trace (gen+1,2)=mean (ObjV);
    FitnV=ranking (ObjV,[2,0],SUBPOP);    % 分配适应度值(Assign fitness values)
    SelCh=select (SEL_F,Chrom,FitnV, GGAP,SUBPOP);            % 选择
    SelCh=recombin (XOV_F, SelCh,XOVR,SUBPOP);               % 重组
    SelCh=mutate (MUT_F,SelCh,FieldDR,[MUTR],SUBPOP) ;      % 变异
    ObjVOff=feval (OBJ_F,SelCh);                % 计算目标函数值
    [Chrom, ObjV]=reins (Chrom, SelCh, SUBPOP,[1 INSR], ObjV, ObjVOff);    % 替代
    gen=gen+1;
end
[Y,I]=min (ObjV);
subplot (211);
plot (Chrom (I,:));
hold on;
plot (Chrom (I,:),'.');grid                 % 最优控制向量分布图
legend ('最优控制向量')
subplot (212);                              % 遗传算法性能跟踪分布图
plot (trace (:,1));
hold on;plot (trace (:,2),'-.');grid
legend ('解的变化','种群均值的变化');xlabel ('迭代次数')
```

遗传算法的显示结果如下：

(1) 经过 100 次迭代后的优化解的目标函数值及性能跟踪如图 7.27 所示。此时，$\min f(x, u)=26\ 503$。

(2) 经过 1000 次迭代后的优化解的目标函数值及性能跟踪如图 7.28 所示。此时，

$\min f(x, u) = 17\ 259$。

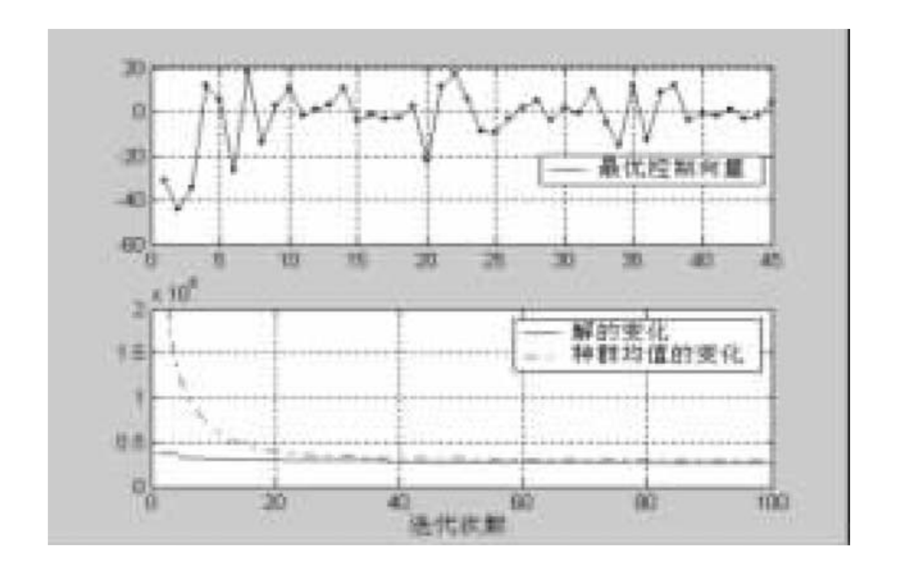



图 7.27　经过 100 次迭代后的优化解的目标函数值及性能跟踪

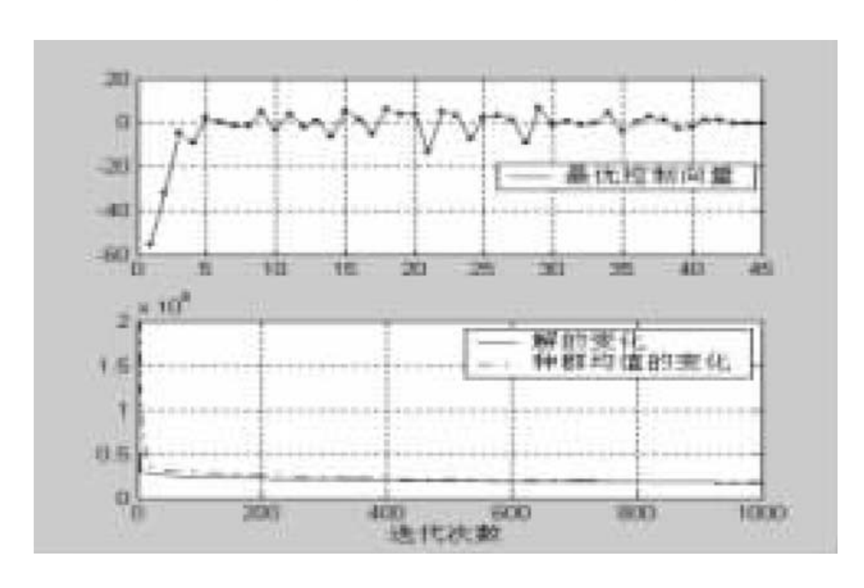



图 7.28　经过 1000 次迭代后的优化解的目标函数值及性能跟踪

（3）经过 2000 次迭代后的优化解的目标函数值及性能跟踪如图 7.29 所示。此时，$\min f(x, u) = 16\ 337$。

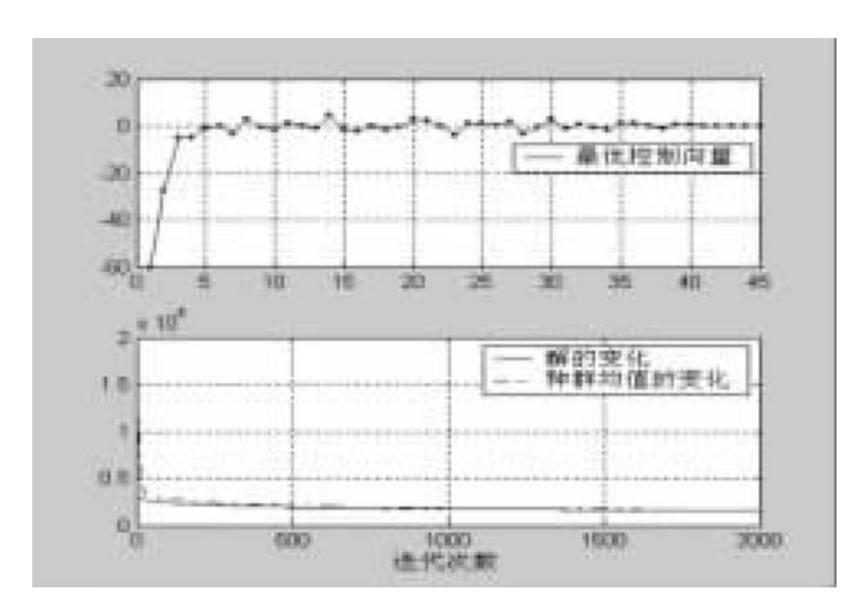



图 7.29　经过 2000 次迭代后的优化解的目标函数值及性能跟踪

# 7.7 目标分配问题

目标分配问题描述为：$m$ 个地空导弹火力单元对 $n$ 批空袭目标进行目标分配。假设进行目标分配之前，各批目标的威胁程度与各火力单元对各批目标的射击有利程度已经经过评估和排序。第 $j$ 批目标的威胁程度评估值为 $w_j$，第 $i$ 个火力单元对第 $j$ 批目标射击有利程度估计值为 $p_{ij}$，令各火力单元对各批目标进行拦击的效益值为 $c_{ij}=w_j \cdot P_{ij}$，其中 $c_{ij}$ 表示对某批目标进行拦击我方获益大小程度。目标分配的目的是满足目标分配的基本原则，追求总体效益最佳，即求 $\max(\sum_{j=1}^{n} c_{ij})$。

染色体采用十进制编码，染色体的长度由按目标批次编号顺序排列的火力单元分配编号组成，表示一种可能的分配方案。

下面为目标分配的最优问题的 MATLAB 代码。

```
 function [eval]=targetalloc (chrom)          % 目标函数
[m,n]=size (chrom);
% 射击有利程度估计值
p=[.87 .52 .11 .78 .72 .69 .94 .72 .36 .28 .27 .74 .24 .78 .45;...
    .87 .52 .11 .78 .72 .69 .94 .72 .36 .28 .27 .74 .24 .78 .45;...
    .87 .52 .11 .78 .72 .69 .94 .72 .36 .28 .27 .74 .24 .78 .45;...
    .87 .52 .11 .78 .72 .69 .94 .72 .36 .28 .27 .74 .24 .78 .45;...
    .87 .52 .11 .78 .72 .69 .94 .72 .36 .28 .27 .74 .24 .78 .45;...
    .87 .52 .11 .78 .72 .69 .94 .72 .36 .28 .27 .74 .24 .78 .45;...
    .62 .87 .70 .22 .80 .42 .43 .90 .13 .95 .18 .19 .12 .61 .35;...
    .48 .20 .42 .16 .43 .58 .69 .03 .34 .72 .15 .24 .29 .30 .75 ];
w=[.47 .97 .76 .62 .48 .77 .33 .74 .54 .65 .43 .35 .63 .66 .57];      % 威胁程度评估值
for i=1:m
    for j=1:15
        chrom (i,j)=p (chrom (i,j),j);
    end;
end
eval=chrom * w';
% 定义遗传算法参数
NIND=40;                         % 个体数目(Number of individuals)
MAXGEN=50;                       % 最大遗传代数(Maximum number of generations)
GGAP=0.9;                        % 代沟(Generation gap)
trace=zeros (MAXGEN,2);          % 遗传算法性能跟踪初始值
BaseV= crtbase (15,8);
Chrom=crtbp (NIND, BaseV)+ones (NIND,15);       % 初始种群
gen=0;
ObjV=taretalloc (Chrom);         % 计算初始种群函数值
while gen < MAXGEN,
    FitnV=ranking (-ObjV);       % 分配适应度值(Assign fitness values)
```

```
    SelCh=select ('sus', Chrom,FitnV, GGAP);                    % 选择
    SelCh=recombin ('xovsp',SelCh,0.7);                          % 重组
    f=rep ([1; 8],[1,15]);
    SelCh=mutbga (SelCh,f);SelCh=fix (SelCh);                    % 变异
    ObjVSel=targetalloc (SelCh);                                 % 计算子代目标函数值
    [Chrom ObjV]=reins (Chrom,SelCh,1,1,ObjV,ObjVSel);           % 重插入
    gen=gen+1;
    trace (gen,1)=max (ObjV);                                    % 遗传算法性能跟踪
    trace (gen,2)=sum (ObjV)/length (ObjV);
end
[Y,I]=max (ObjV);Chrom (I,:),Y                                   % 最优解及其目标函数值
plot (trace (:,1),'-.'); hold on;
plot (trace (:,2)); grid
legend ('解的变化','种群均值的变化')
```

目标函数设计为 function [eval]=targetalloc (chrom)。其中，chrom 为染色体，在函数体中 p 为 8 个火力单元对 15 批目标的射击有利程度评估值。w 为 15 批目标的威胁程度评估值。遗传算法中，个体的数量 NIND 被设置为 40，最大遗传代数 MAXGEN=50，染色体长度为 15，使用代沟 GGAP=0.9，使用基于适应度的重插入最适应的个体总是被连续传播到下一代。区域描述器 BaseV= crtbase (15,8)描述染色体的表示和解释，染色体采用十进制编码，一个初始种群被函数 crtbp 创建，随后产生一个矩阵 Chrom，它由 NIND 个长度为 15 的十进制串构成。Chrom=crtbp (NIND，BaseV)处加 ones (NIND,15)的目的是保证矩阵处理中的行、列序号不为零。

经过 50 次遗传迭代后，目标分配方案见表 7.3。

表 **7.3** 目标分配方案

| 目标编号 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 分配结果 | 3 | 7 | 7 | 2 | 7 | 4 | 6 | 7 | 1 | 7 | 5 | 3 | 8 | 1 | 8 |

与此方案对应的总收益值为 6.4719。图 7.30 为经过 50 次迭代后的优化解的目标函数值及性能跟踪。

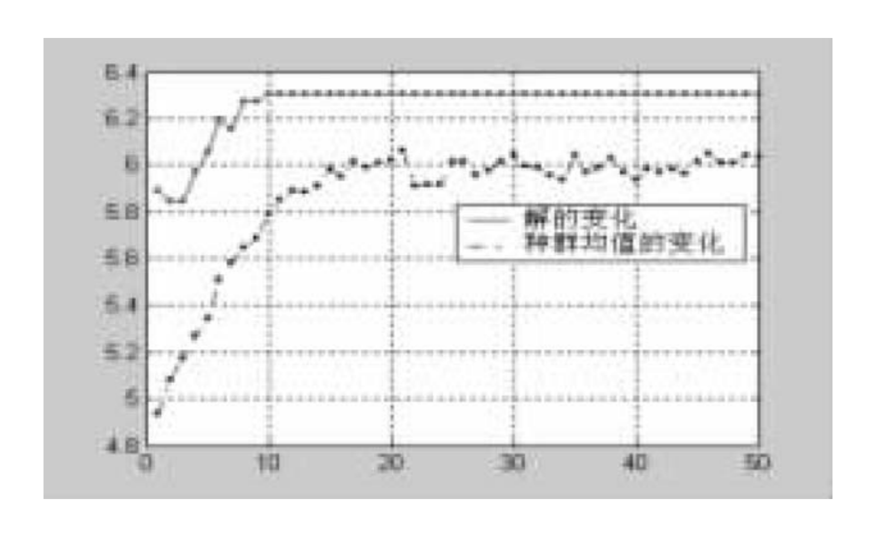



图 7.30 经过 50 次迭代后的优化解的目标函数值及性能跟踪

# 7.8 双积分的优化问题

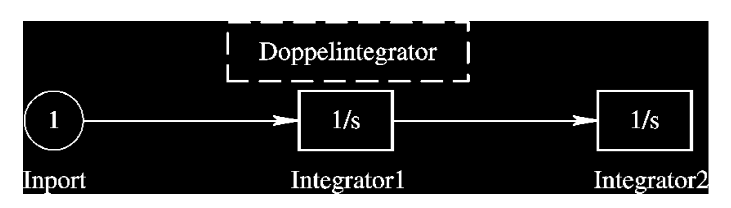
双积分问题的状态方程为

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = x_1 \\ y = x_2 \end{cases}$$

时间范围为：$0 \leqslant t \leqslant 1$。初始条件为：$x_1(0)=0$；$x_2(0)=-1$。终止条件为：$x_1(0)=0$；$x_2(0)=0$。目标函数为

$$\min f(u) = \int_0^1 u(t)^2 \mathrm{d}t$$

双积分问题的 Simulink 模型如下：




一个实现本目标函数的 M 文件 objdopi 包含在 GA 工具箱软件中。

下面为双积分的优化问题的 MATLAB 代码。

```
 % 定义遗传算法参数
Dim=20;              % 变量维数
NIND =20;            % 个体数目(Number of individuals)
Preci=20;            % 变量的二进制位数(Precision of variables)
MAXGEN=100;          % 最大遗传代数(Maximum number of generations)
GGAP=0.8;            % 代沟(Generation gap)
SEL_F='sus';         % 选择函数名
XOV_F ='xovsp';      % 重组函数名
MUT_F ='mut';        % 变异函数名
OBJ_F ='objdopi';    % 目标函数名
FieldDR=feval (OBJ_F,[ ],1);                      % 计算目标函数值
% 建立区域描述器(Build field descriptor)
FieldDD=[rep ([Preci],[1,Dim]);FieldDR;rep ([1;0;1;1],[1,Dim])];
Chrom=crtbp (NIND,Dim * Preci);                   % 创建初始种群
gen=0;
Best=NaN * ones (MAXGEN,1);                       % 最优解初值
while gen < MAXGEN,                               % 最大循环次数
    ObjV=feval (OBJ_F,bs2rv (Chrom,FieldDD));     % 计算目标函数值
    Best (gen+1)=min (ObjV);                      % 最优解
    plot (log10 (Best),'bo');
    FitnV=ranking (ObjV);                         % 分配适应度值(Assign fitness values)
    SelCh=select (SEL_F,Chrom,FitnV, GGAP);       % 选择
    SelCh=recombin (XOV_F, SelCh);                % 重组
```

```
    SelCh=mutate (MUT_F,SelCh);                     % 变异
    Chrom=reins (Chrom, SelCh);                     % 重插入
    gen=gen+1;
end
grid;
xlabel ('迭代次数');ylabel ('目标函数值(取对数)') ;
```

遗传算法的运行结果如下：

（1）经过 50 次迭代后的运行结果如图 7.31 所示。

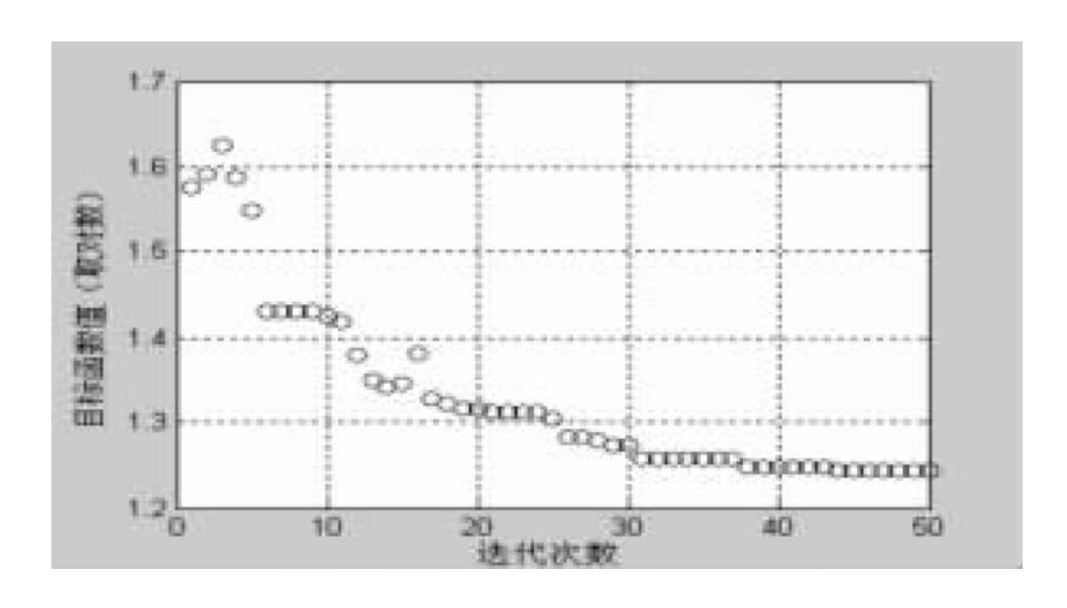



图 7.31　经过 50 次迭代后的运行结果

（2）经过 100 次迭代后的运行结果如图 7.32 所示。

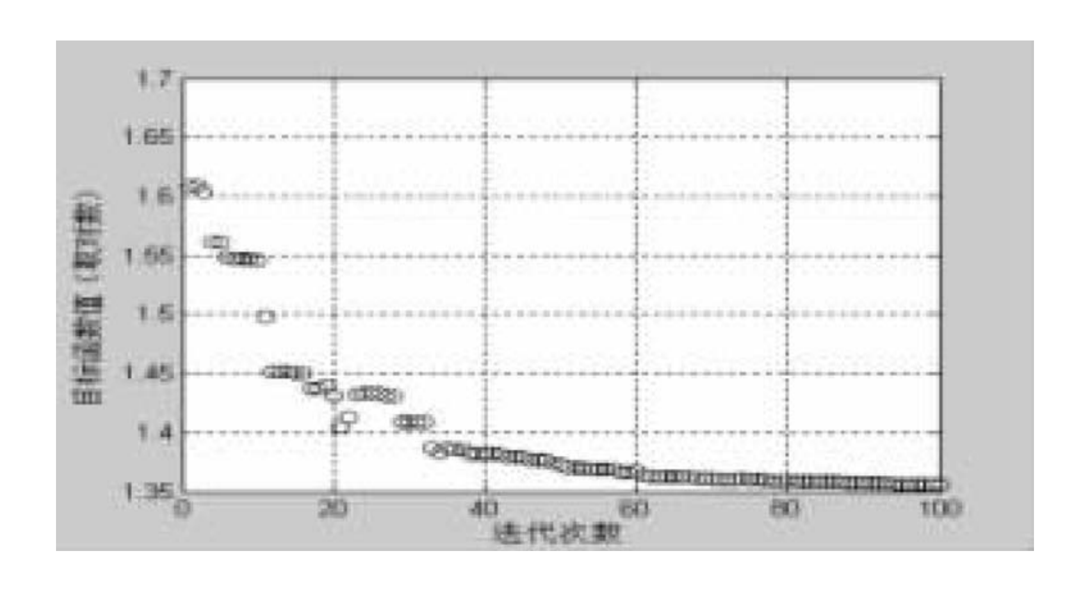



图 7.32　经过 100 次迭代后的运行结果

# 7.9　雷达目标识别问题

复杂目标的高分辨雷达回波信号在时域上出现多个峰值，这些峰值对应目标径向上的多个强散射中心。但是，当空中目标的位置发生变化时，目标的强散射点相对于雷达视角发生变化，因此，从数据库中寻找与待识别目标模式相匹配的模式的工作量很大，传统的

匹配方法已不能满足需要。本节利用遗传算法进行目标识别，具体步骤如下：

（1）染色体编码与解码：假设数据库中已知目标类型有 $m$ 种，每种类型目标在雷达视角 $\alpha\sim\beta$ 范围内的雷达回波信号 $n$ 个序列。对于两参数目标类型 $i\leqslant m$ 和雷达视角编码 $j\leqslant n$，设 $k_1$ 为参数 $i$ 的二进制编码长度，$k_2$ 为参数 $j$ 的二进制编码长度，定义遗传算法的个体 $I_k$ 基因型 $G_k$ 为

$$G_k = \underbrace{011\cdots0101}_{k_1}\ \underbrace{101\cdots101}_{k_2}$$

则编码总长度为 $k_1+k_2$，即个体 $I_k$ 基因型 $G_k$ 的前 $k_1$ 位的十进制解码为 $i$，后 $k_2$ 位的十进制解码为 $j$。

（2）产生初始种群：一个个体由串长为 $k_1+k_2$ 的随机产生的二进制串组成染色体的基因码，可以产生一定数目的个体组成种群，设置群体大小。

（3）个体适应度的检测评估：设 $X(i,j)$为第 $i$ 种目标在第 $j$ 个雷达视角的高分辨雷达目标回波信号序列，对应的个体为 $I_k$。待识别目标的高分辨雷达目标回波信号序列为 $X$。个体 $I_k$ 的适应度函数可定义为

$$f(I_k) = \frac{|X(i,j)\otimes X|}{\sqrt{|X(i,j)\otimes X(i,j)\|X\otimes X|}}$$

式中，$|X\otimes Y|$为序列 $X$ 和序列 $Y$ 的相关系数。

（4）遗传算子：选择运算使用比例选择算子，交叉运算使用单点交叉算子，变异运算使用基本位变异算子或均匀变异算子。每个个体选择概率为

$$p_s = \frac{f(I_k)}{\sum_{k=1} f(I_k)}$$

设置遗传算法的终止进化代数、交叉概率和变异概率，对群体进行操作。

下面为雷达目标识别问题的 MATLAB 代码。

```
function coef=corrcoef1(x,y)    % 计算 x 和 y 的相关系数函数，x 为单序列，y 为复序列
[m,n]=size(y);
coef=zeros(m,1);
for i=1:m
    coef(i)=sum(abs(conv(x,y(i,:))))/sqrt( sum(abs(conv(x,x))) * sum(abs(conv(y(i,:),
            y(i,:)))));
end
function class=recognite(signal,inform);     % 目标识别函数
% signal 为待识别目标的雷达信号；inform 目标识别数据库中的模板信号
NIND=20;                               % 个体数目(Number of individuals)
MAXGEN =200;                           % 最大遗传代数(Maximum number of generations)
Type=10;Angle=20;                      % 目标类型 10bit，雷达视角 20bit
PRECI=Type + Angle;                    % 变量的二进制位数(Precision of variables)
GGAP=0.9;                              % 代沟(Generation gap)
FieldD=[PRECI;1;NIND;1;0;1;1];         % 区域描述器(Build field descriptor)
Chrom=crtbp(NIND, PRECI);              % 创建初始种群
gen=0;
```

```
v=bs2rv(Chrom,FieldD);                      % 种群十进制转换
class=round(v);                             % 取整
ObjV=corrcoef1(signal,inform(class,:));     % 计算相关系数
trace=zeros(MAXGEN);                        % 性能跟踪
while gen < MAXGEN,
    FitnV=ranking(-ObjV);                   % 分配适应度值(Assign fitness values)
    SelCh=select('sus', Chrom, FitnV, GGAP);                    % 选择
    SelCh=recombin('xovsp',SelCh,0.7);                          % 重组
    SelCh=mut(SelCh);                                           % 变异
    v=bs2rv(SelCh,FieldD);
    class=round(v);
    ObjVSel=corrcoef1(signal,inform(class,:));                  % 计算目标函数相关系数
    [Chrom ObjV]=reins(Chrom,SelCh,1,1,ObjV,ObjVSel);           % 重插入
    gen=gen+1;
    trace(gen)=max(ObjV);
end
% 输出最优解及其序号，并在目标函数图像中标出，Y为最优解，I为种群的序号
[Y,I]=max(ObjV);
Type=round(bs2rv(Chrom(I,1:Type),FieldD));                      % 目标类型编号
Angle=round(bs2rv(Chrom(I,Type+1:Angle),FieldD));               % 雷达视角编号
plot(trace);hold on;
plot(trace,'.'); grid;
```

实例：实验数据选自毫米波步进频率信号，带宽为1 GHz，调频步长为2 MHz。识别目标为轰炸机、歼击机和直升机的缩比模型。对三类飞机鼻锥方向±10°方位角(角度间隔为1°)变化范围内的雷达回波进行截取、去均值、归一化等预处理后，作为数据库中的数据。待识别目标的雷达回波数据为三类飞机的雷达回波加高斯白噪声。图7.33为遗传算法搜索最优解的过程。

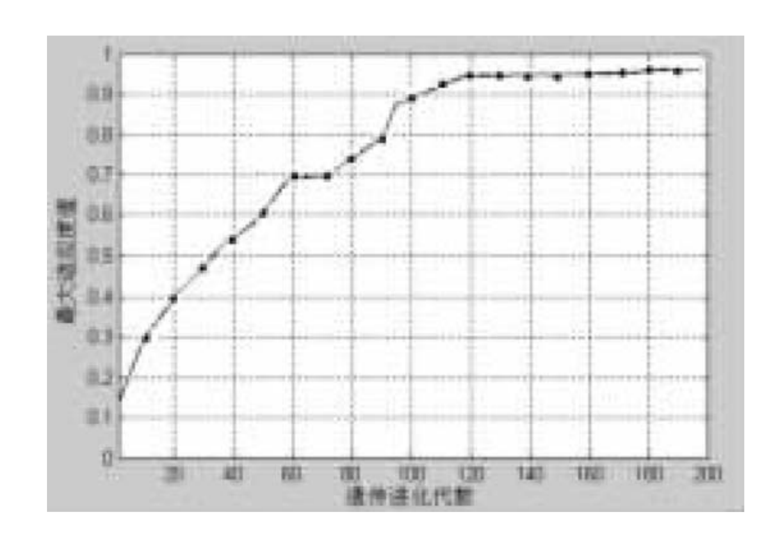


图7.33 遗传算法搜索最优解的过程

# 7.10 图像分割问题

利用遗传算法进行图像分割的基本思想是：把图像中的像素按灰度值用阈值 $M$ 分成两类图像，一类为目标图像 $C_1$，另一类为背景图像 $C_2$。图像 $C_1$ 由灰度值在 $0 \sim M$ 之间的像素组成，图像 $C_2$ 由灰度值在 $M+1 \sim L-1$（$L$ 为图像的灰度级数）之间的像素组成。

本节处理图像为 256 灰度级，将灰度分割阈值编码为一个 8 位 0、1 二进制码串。适应度函数选为 $f(M)$：

$$f(M) = W_1(M) \cdot W_2(M) \cdot [U_1(M) - U_2(M)]^2$$

其中，$W_1(M)$为目标图像 $C_1$ 中所包含的像素数；$W_2(M)$为目标图像 $C_2$ 中所包含的像素数；$U_1(M)$为 $C_1$ 中所有像素数的平均灰度值；$U_2(M)$为 $C_2$ 中所有像素数的平均灰度值。

下面为雷达目标识别问题的 MATLAB 代码。

```
function f=target(T,M)                % 适应度函数，T 为待处理图像，M 为阈值序列
[U,V]=size(T);
W=length(M);
f=zeros(W,1);
for k=1:W
I=0;s1=0; J=0; s2=0;                  % 统计目标图像和背景图像的像素数及像素之和
for i=1:U
    for j=1:V
        if T(i,j)<=M(k)
            s1=s1+T(i,j);I=I+1;
        end
        if T(i,j)>M(k)
            s2=s2+T(i,j);J=J+1;
        end
    end
end
p1=s1/I;p2=s2/J;                      % 计算目标图像与背景图像的像素平均值
f(k)=I*J*(p1-p2)*(p1-p2)/(256*256);        % 计算适应度函数值
end
load woamn                            % 调用 MATLAB 中 Woman 图像灰度值
figure(1);                            % 画图
image(X);colormap(map);
NIND=40;                              % 个体数目(Number of individuals)
MAXGEN =50;                           % 最大遗传代数(Maximum number of generations)
PRECI=8;                              % 变量的二进制位数(Precision of variables)
GGAP=0.9;                             % 代沟(Generation gap)
FieldD=[8;1;256;1;0;1;1];             % 建立区域描述器(Build field descriptor)
Chrom=crtbp(NIND, PRECI);             % 创建初始种群
gen=0;
```

```
phen=bs2rv(Chrom,FieldD);              % 初始种群十进制转换
ObjV=target(X,phen);                   % 计算种群适应度值
while gen < MAXGEN,                    % 代沟(Generation gap)
    FitnV=ranking(-ObjV);              % 分配适应度值(Assign fitness values)
    SelCh=select('sus', Chrom, FitnV, GGAP);    % 选择
    SelCh=recombin('xovsp',SelCh,0.7);          % 重组
    SelCh=mut(SelCh);                           % 变异
    phenSel=bs2rv(SelCh,FieldD);                % 子代十进制转换
    ObjVSel=target(X,phenSel);
    [Chrom ObjV]=reins(Chrom,SelCh,1,1,ObjV,ObjVSel);   % 重插入
    gen=gen+1;
end
[Y,I]=max(ObjV);
M=bs2rv(Chrom(I,:),FieldD) ;                    % 估计阈值
[m,n]=size(X);
for i=1:m
    for j=1:n
        if X(i,j)>M
            X(i,j)=256;                         % 灰度值为 256 时是白色
        end
    end
end
figure(2)                                       % 画出分割后目标图像
image(X);colormap(map);
```

图 7.34 为 MATLAB 中的 Woman 图像。图 7.35 为经过 50 次迭代后的分割图像(阈值为 117)。



图 7.34　Woman 原始图像(256×256)




图 7.35　Woman 分割后的图像(阈值为 117)

# 7.11　一些测试函数对应的优化问题

下面给出工具箱提供的一些测试函数的遗传算法的运行结果(问题的 MATLAB 代码可参考前面例子的代码)。

## 7.11.1　轴并行超球体的最小值问题

一个实现本目标函数的 M 文件 objfunla 包含在 GA 工具箱软件中。

变量维数 Dim＝10，变量的范围为－512～512，选取个体数目 NIND＝40，迭代次数 MAXGEN 取为 100、200、500，代沟 GGAP 取为 0.9。迭代 100 次、200 次、500 次目标函数最小值分别为 3141.2、5.7440、$1.3113\times10^{-5}$，最小值对应的 10 个变量值见表 7.4。目标函数的理论最小值为 0。跟踪性能如图 7.36、图 7.37 和图 7.38 所示。

**表 7.4　迭代 100 次、200 次、500 次的目标函数最小值对应的 10 个变量值**

| 迭代次数 | 变量值 | | | | | | | | | | 备注 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 100 | 15.3706 | －3.4136 | －4.9595 | 0.6216 | 16.0913 | 13.7593 | 6.3384 | 0.2026 | 2.8608 | 1.4341 | |
| 200 | 0.1421 | 1.0708 | 0.2427 | 0.0298 | －0.4663 | －0.3452 | 0.0562 | 0.0278 | －0.3081 | 0.2378 | |
| 500 | －0.4883 | －0.4883 | 0.4883 | －0.4883 | 0.4883 | －0.4883 | －0.4883 | －0.4883 | －0.4883 | 0.4883 | $\times10^{-3}$ |

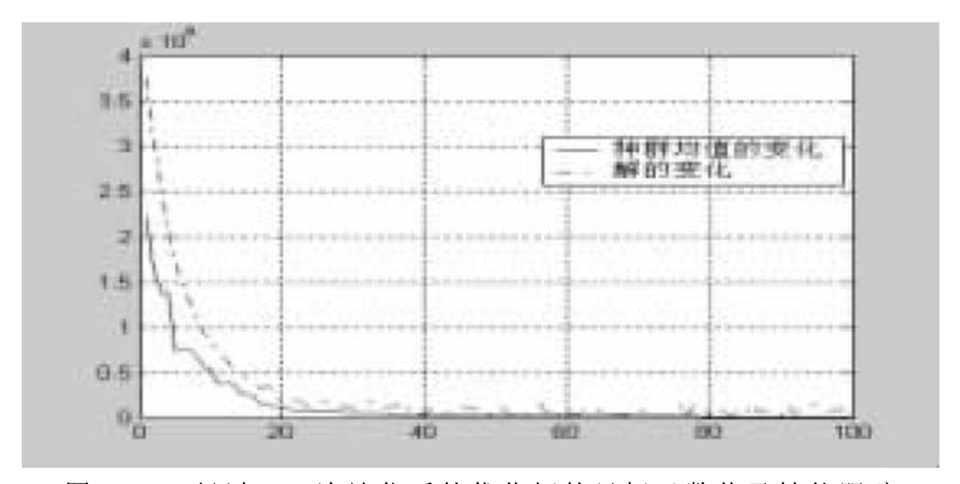



图 7.36　经过 100 次迭代后的优化解的目标函数值及性能跟踪

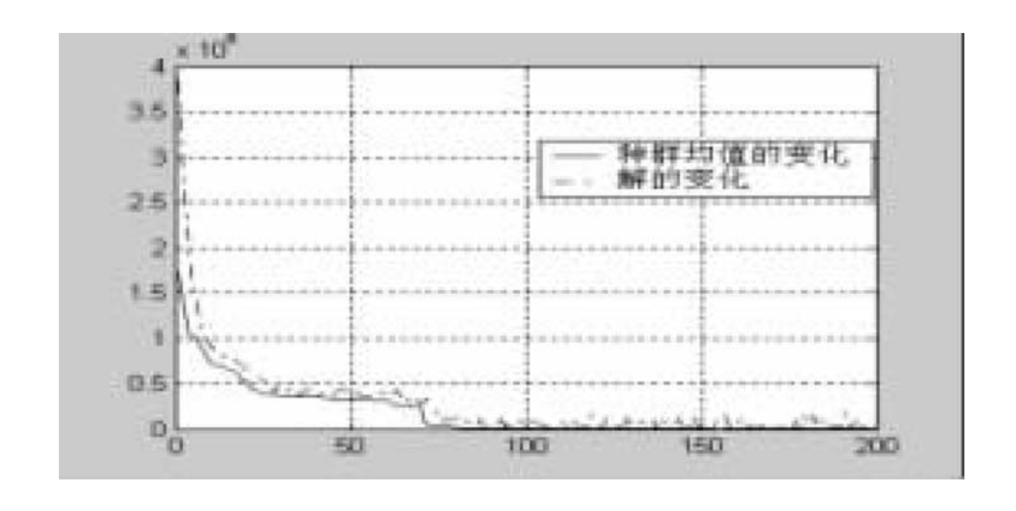



图 7.37　经过 200 次迭代后的优化解的目标函数值及性能跟踪

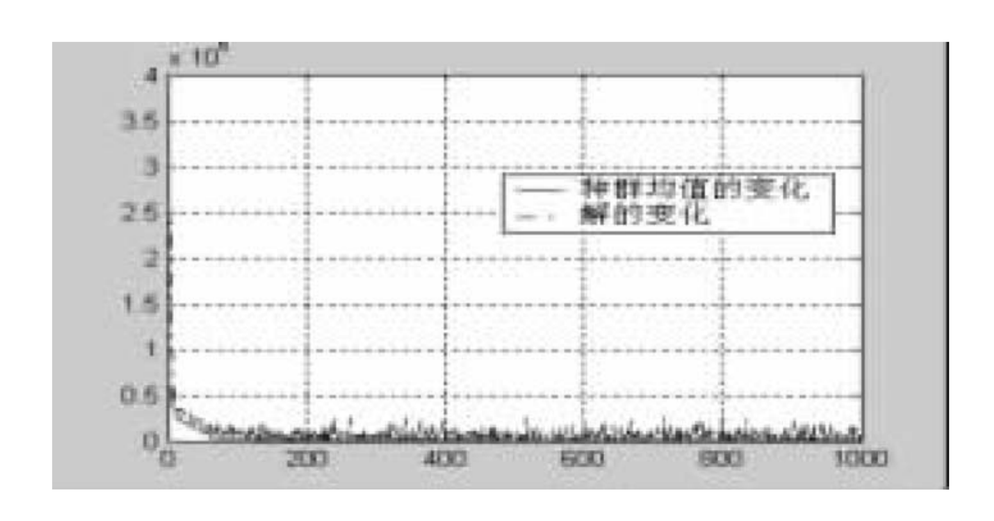



图 7.38　经过 500 次迭代后的优化解的目标函数值及性能跟踪

### 7.11.2　旋转超球体的最小值问题

一个实现本目标函数的 M 文件 objfunlb 包含在 GA 工具箱软件中。

变量维数 Dim＝10，变量的范围为－65～65，选取 NIND＝40，MAXGEN＝100、500、1000；GGAP＝0.9。表 7.5 分别为迭代 100 次、500 次、1000 次的目标函数最小值对应的变量值，对应的最小值分别为 483.2808、48.8648、24.3775。目标函数理论值为 0。跟踪性能如图 7.39、图 7.40 和图 7.41 所示。

**表 7.5　迭代 100 次、500 次、1000 次的目标函数最小值对应的 10 个变量值**

| 迭代次数 | 变量值 | | | | | | | | | | 备注 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 100 | 8.5230 | －8.0729 | 9.4457 | －24.959 | 14.1763 | 1.6832 | 2.1510 | －9.3494 | 10.389 | －8.3015 | |
| 500 | 3.8979 | －4.1485 | －3.9814 | 7.8619 | －3.5510 | －0.7377 | 0.5833 | 1.4464 | －1.6146 | 0.6152 | |
| 1000 | 3.3259 | －5.8697 | 4.9317 | －2.6650 | 0.6560 | －0.4382 | 0.7710 | －1.0381 | 0.5261 | 0.3165 | $\times 10^{-3}$ |

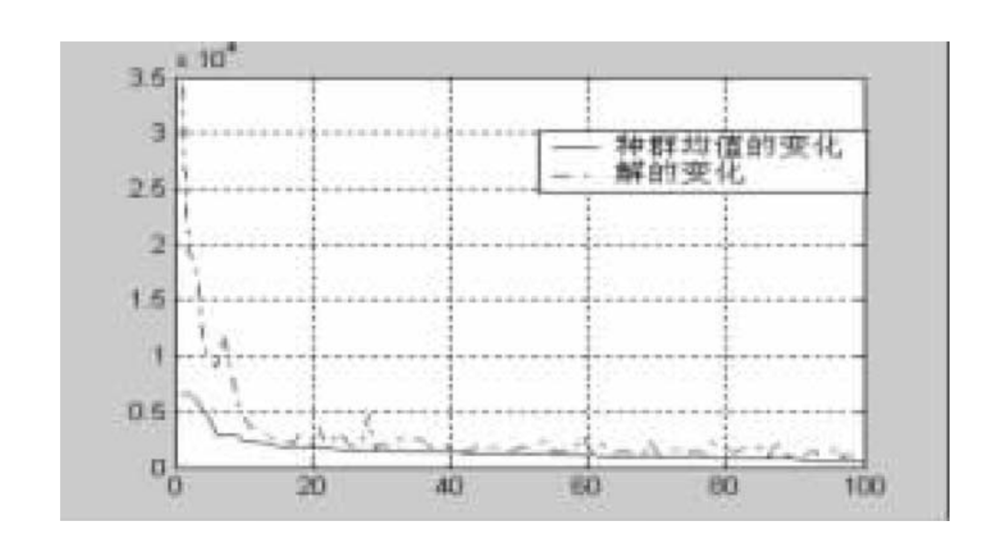



图 7.39　经过 100 次迭代后的优化解的目标函数值及性能跟踪

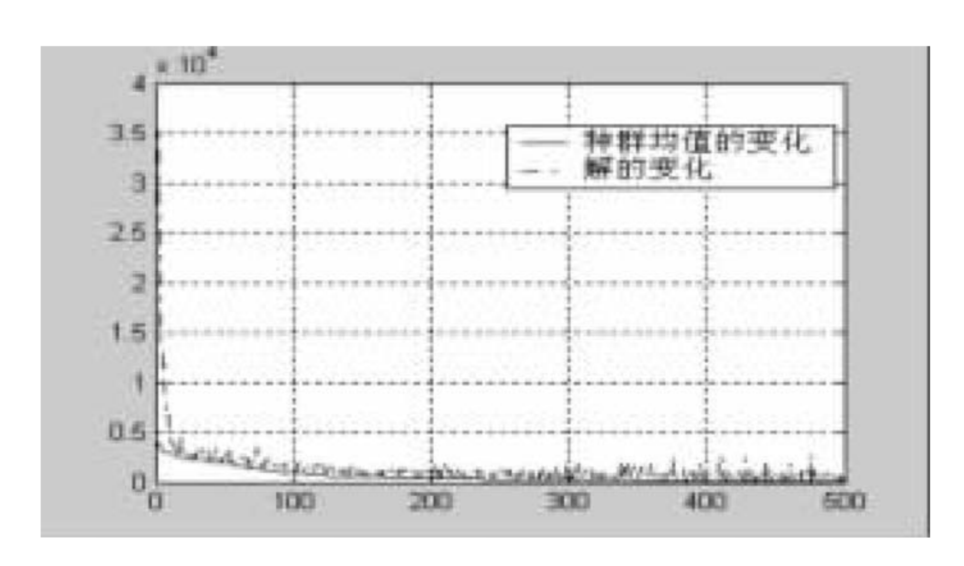



图 7.40　经过 500 次迭代后的优化解的目标函数值及性能跟踪

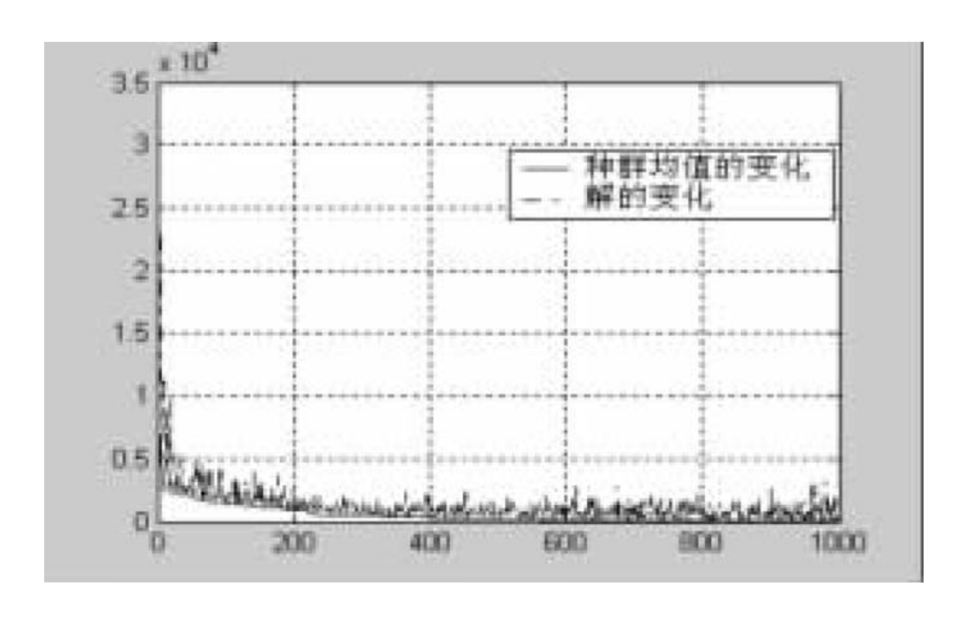



图 7.41　经过 1000 次迭代后的优化解的目标函数值及性能跟踪

## 7.11.3　Rosenbrock's Valley 最小值问题

一个实现本目标函数的 M 文件 objfun2 包含在 GA 工具箱软件中。

变量维数 Dim＝2，变量的范围为－2～2，选取 NIND＝40，GGAP＝0.9。表 7.6 为迭代的最优解和对应的目标函数值。理论值为：变量值为[0,0]，目标函数值为 0。跟踪性能如图 7.42 所示。

表 **7.6** 迭代 **50** 次、**100** 次、**200** 次、**500** 次和 **1000** 次的最优解及对应的目标函数值

| 迭代次数 | 50 | | 100 | | 200 | | 500 | | 1000 | |
|---|---|---|---|---|---|---|---|---|---|---|
| 变量值 | 0.8347 | 0.6959 | 1.1193 | 1.2523 | 0.8970 | 0.8040 | 1.0836 | 1.1741 | 1.0000 | 1.0001 |
| 目标函数值 | 0.0274 | | 0.0142 | | 0.0106 | | 0.0070 | | $1.2049\times10^{-9}$ | |

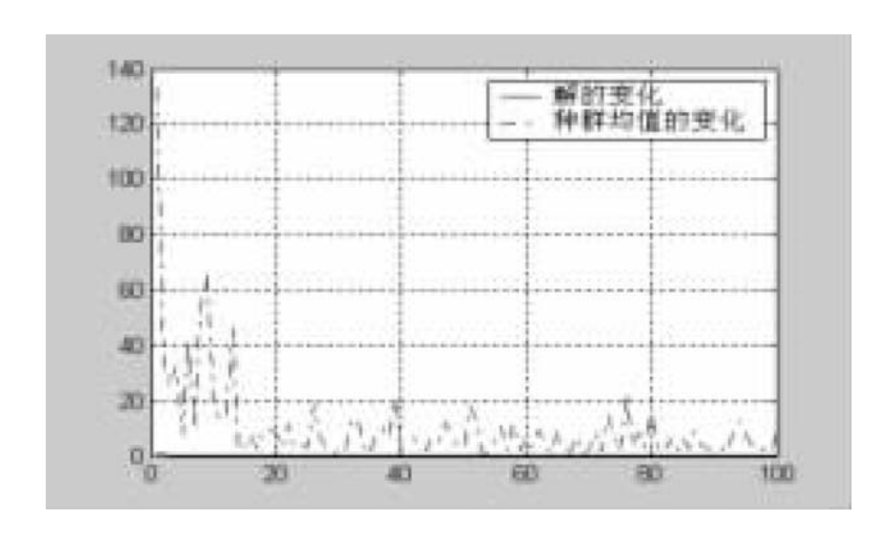



图 7.42 经过 100 次迭代后的优化解的目标函数值及性能跟踪

## 7.11.4 Rastrigin 函数的最小值问题

一个实现本目标函数的 M 文件 objfun6 包含在 GA 工具箱软件中。

变量维数 Dim＝20，变量的范围为－5.12～5.12，选取 NIND＝40，GGAP＝0.9。目标函数理论值为 0。表 7.7 为迭代次数及对应的目标函数最小值。迭代次数为 1000 时目标函数取最小值，对应的 20 个变量的值见表 7.8，也可见图 7.43。跟踪性能见图 7.44。

表 **7.7** 迭代次数及对应的最小值

| 迭代次数 | 50 | 100 | 200 | 500 | 1000 |
|---|---|---|---|---|---|
| 最小值 | 66.5403 | 34.8589 | 33.0944 | 27.8595 | 23.8798 |

表 **7.8** 迭代次数为 **1000** 时 **20** 个变量的值

| 序号 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 变量值 | 0.9950 | －0.0000 | －0.9950 | 0.0000 | －0.9950 | 0.9950 | －0.9950 | －0.0000 | 0.9950 | －0.9950 |
| 序号 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 变量值 | －0.9949 | 0.0000 | 0.0000 | －0.0000 | －1.9899 | －0.9949 | 0.9950 | 1.9899 | 0.9950 | －0.9950 |

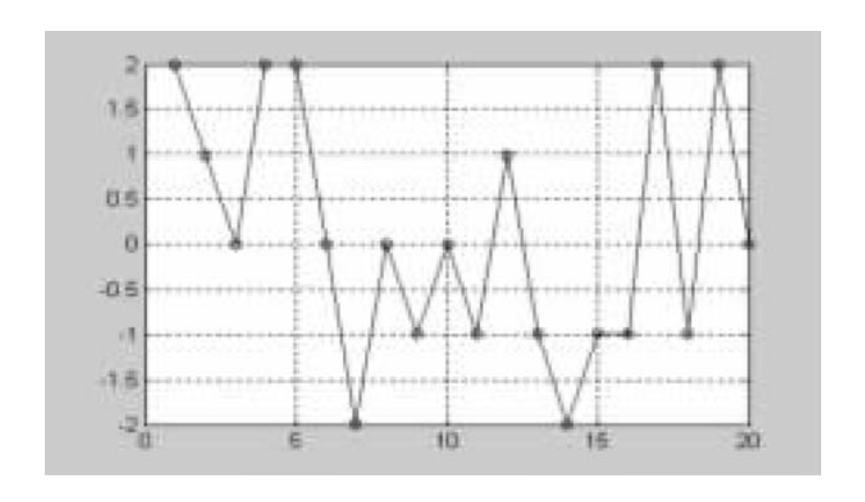



图 7.43　经过 1000 次迭代后的最优解

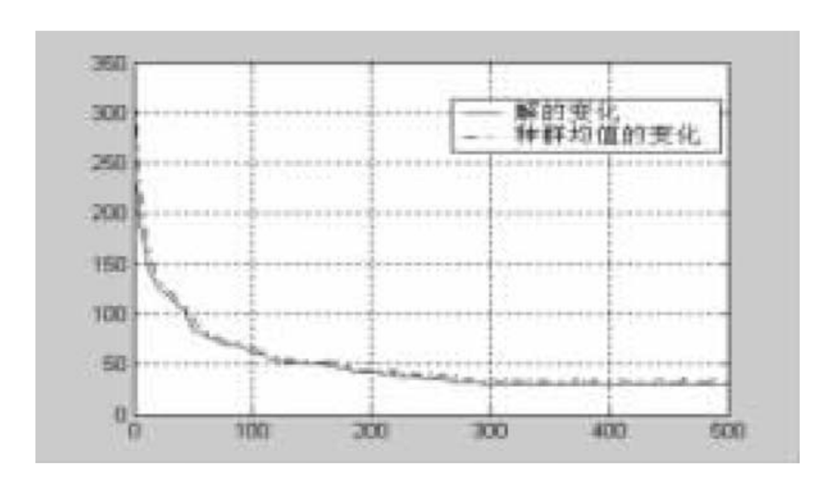



图 7.44　经过 500 次迭代后的优化解的目标函数值及性能跟踪

### 7.11.5　Schwefel 函数的最小值问题

一个实现本目标函数的 M 文件 objfun7 包含在 GA 工具箱软件中。

变量维数为 20，变量范围为－500～500，选取 NIND＝40，GGAP＝0.9。表 7.9 为迭代次数及对应的最小值。理论最小值为－8379.7。跟踪性能见图 7.45，迭代次数为 1000 时最优解如图 7.46 所示。

表 **7.9**　迭代次数及对应的最小值

| 迭代次数 | 50 | 100 | 200 | 500 | 1000 |
|---|---|---|---|---|---|
| 最小值 | －6228.4 | －6416.4 | －7730.0 | －7932.6 | －8255.8 |

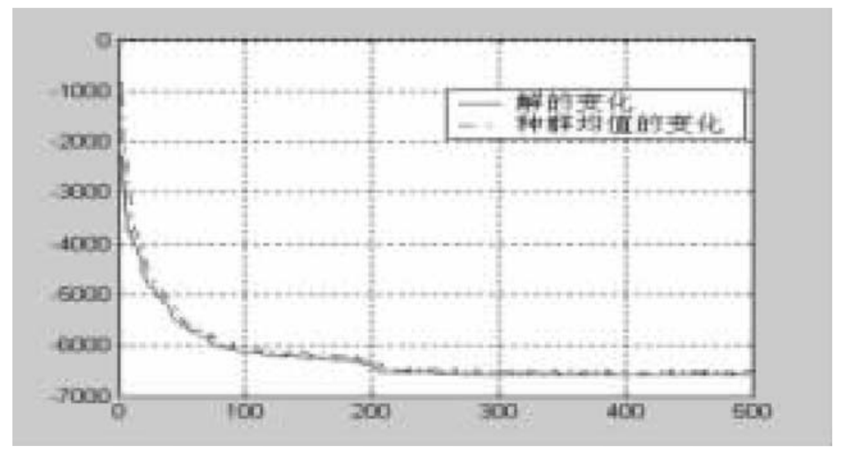



图 7.45　经过 500 次迭代后的最优解及性能跟踪

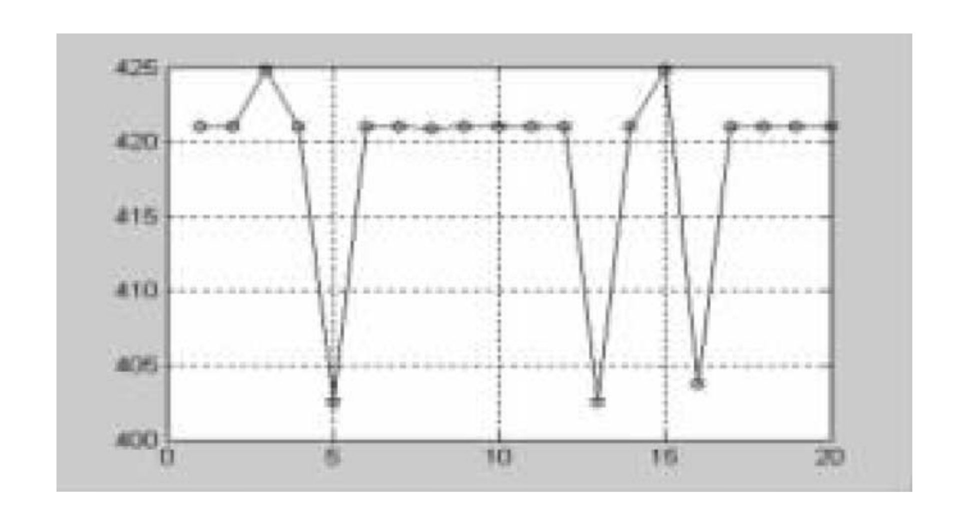


图 7.46　经过 1000 次迭代后的最优解

### 7.11.6　Griewangk 函数的最小值问题

一个实现本目标函数的 M 文件 objfun8 包含在 GA 工具箱软件中。

变量维数为 10，变量范围为－600～600，选取 NIND＝40，GGAP＝0.9。表 7.10 为迭代次数及对应的最小值。理论值为 0。跟踪性能如图 7.47 所示。迭代 200 次和 500 次后的最优解对应的变量值见表 7.11。

表 **7.10**　迭代次数及对应的最小值

| 迭代次数 | 50 | 100 | 200 | 500 |
|---|---|---|---|---|
| 最小值 | 2. 9075 | 1. 2277 | 0. 3920 | 0. 0369 |

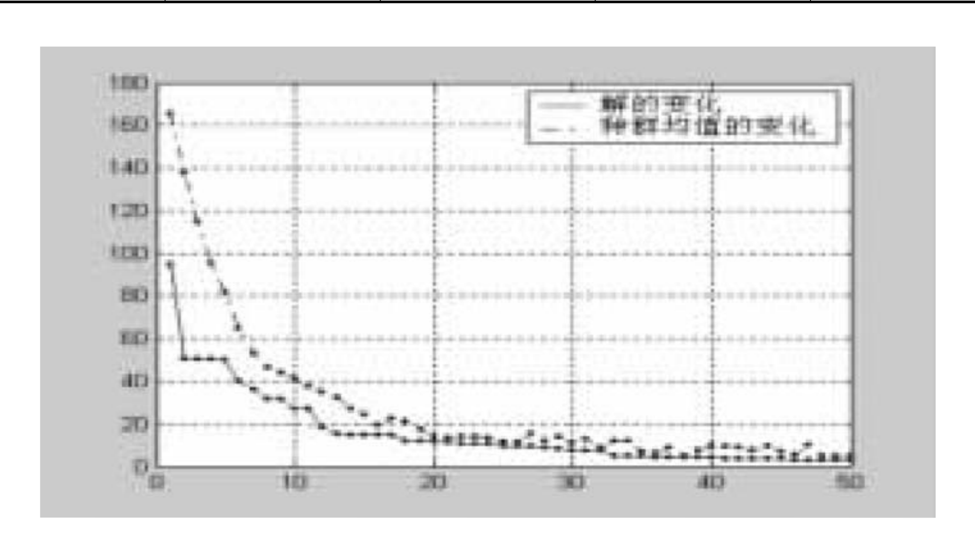



图 7.47　经过 50 次迭代后的最优解及性能跟踪

表 **7.11**　迭代次数为 **200** 和 **500** 后的最优解对应的变量值

| 迭代次数 | 变　量　值 | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 200 | 6.2834 | －17.7767 | 5.4033 | 18.8158 | 0.0647 | 7.9153 | －16.5739 | －8.8228 | －0.0154 | －19.7668 |
| 500 | 3.1397 | －0.0006 | 5.4331 | －0.0006 | －7.0078 | 7.6727 | －0.0006 | 0.0006 | －0.0006 | －0.0006 |

### 7.11.7 不同权的总和最小值问题

一个实现本目标函数的 M 文件 objfun9 包含在 GA 工具箱软件中。

变量维数为 10，变量取值范围为 −1～1，选取 NIND＝40，GGAP＝0.9。表 7.12 为迭代次数及对应的最小值。理论值为 0。跟踪性能如图 7.48 所示，迭代次数为 100 和 200 时的最优解见表 7.13。

表 **7.12** 迭代次数及对应的最小值

| 迭代次数 | 50 | 100 | 200 |
|---|---|---|---|
| 最小值 | 1.0643e−4 | 3.0888e−5 | 1.0175e−10 |

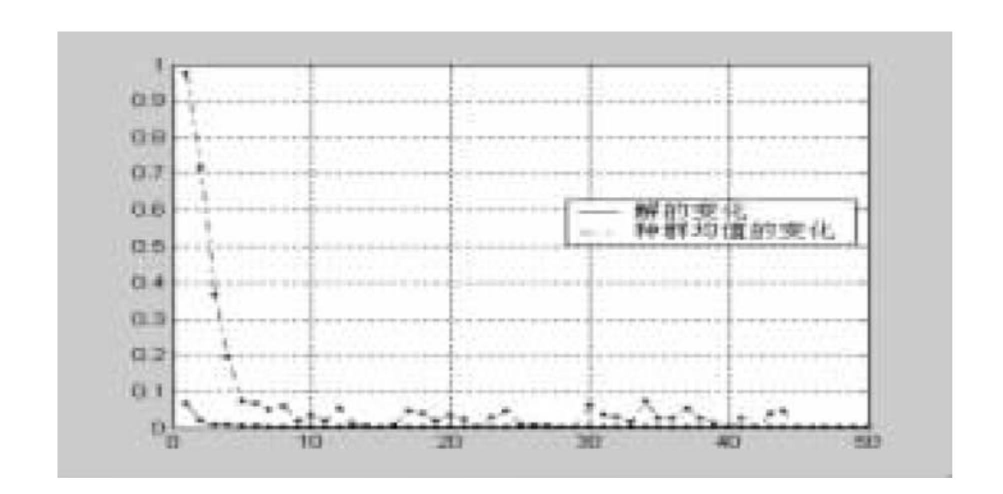



图 7.48 经过 50 次迭代后的最优解及性能跟踪

表 **7.13** **100** 次和 **200** 次迭代后的最优解对应的变量值

| 迭代次数 | 变 量 值 | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 100 | −0.0001 | 0.0011 | −0.0094 | 0.1252 | −0.0033 | −0.0878 | −0.0537 | −0.0403 | 0.0050 | −0.1341 |
| 200 | −0.0000 | 0.0001 | 0.0006 | 0.0098 | 0.0147 | 0.0104 | −0.0124 | −0.0174 | 0.0626 | 0.0012 |

## 7.12 多目标优化问题

下面为一个含有两个优化目标的多目标优化问题。

$$\begin{cases} \min \quad f_1 = \dfrac{x_1^2}{4} + \dfrac{x_2^2}{4} \\ \min \quad f_2 = x_1(1 - x_2) + 10 \\ \text{s.t.} \quad 1 \leqslant x_1 \leqslant 4,\ 1 \leqslant x_2 \leqslant 2 \end{cases}$$

对于该问题，利用权重系数变换法很容易求出 Pareto 最优解。当 $f_1$ 和 $f_2$ 的权重系数都为 0.5 时，经过 50 次迭代后结果为：$x_1=2.0034$，$x_2=2.0000$，$f_1+f_2=10.0001$。跟踪性能如图 7.49 所示。

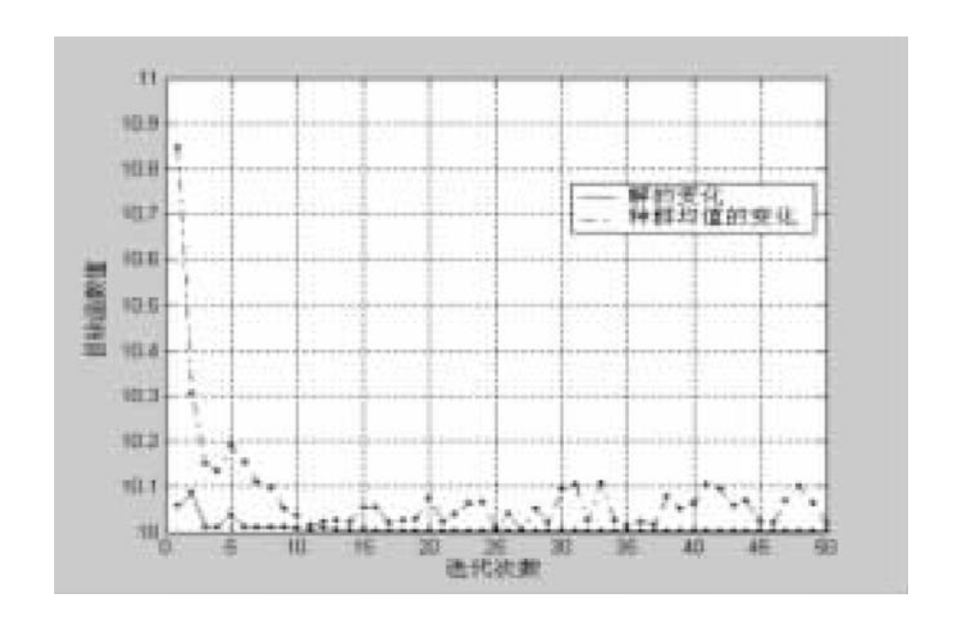



图 7.49　经过 50 次迭代后的最优解及性能跟踪

下面为利用并列选择法求解多目标最优问题的 MATLAB 代码。

```
Function f1=f (x)                          % 第一目标函数
f1=x (:,1). * x (:,1)/4+x (:,2). * x (:,2)/4;
function f2=f (x)                          % 第二目标函数
f2=x (:,1). * (1-x (:,2))+10;
NIND=100;                                  % 个体数目(Number of individuals)
MAXGEN =50;                                % 最大遗传代数(Maximum number of generations)
NVAR=2;                                    % 变量个数
PRECI=20;                                  % 变量的二进制位数(Precision of variables)
GGAP=0.9;                                  % 代沟(Generation gap)
trace1=[ ];trace2=[ ];trace3=[ ];          % 性能跟踪
% 建立区域描述器(Build field descriptor)
FieldD=[rep ([PRECI],[1,NVAR]);[1,1;4,2]; rep ([1;0;1;1],[1,NVAR])];
Chrom=crtbp (NIND, NVAR * PRECI);          % 初始种群
v=bs2rv (Chrom,FieldD);                    % 初始种群十进制转换
gen=1;
while gen < MAXGEN,
    [NIND,N]=size (Chrom);
    M=fix (NIND/2);
    ObjV1=f1 (v (1:M,:));                  % 分组后第一目标函数值
    FitnV1=ranking (ObjV1);                % 分配适应度值(Assign fitness values)
    SelCh1=select ('sus', Chrom (1:M,:), FitnV1, GGAP);         % 选择
    ObjV2=f2 (v (M+1:NIND,:));             % 分组后第二目标函数值
    FitnV2=ranking (ObjV2);
    SelCh2=select ('sus', Chrom ((M+1):NIND,:), FitnV2, GGAP);  % 选择
    SelCh=[SelCh1;SelCh2];                 % 合并
    SelCh=recombin ('xovsp',SelCh,0.7);    % 重组
    Chrom=mut (SelCh);                     % 变异
    v=bs2rv (Chrom,FieldD);
```

```
        trace1 (gen,1)=min (f1 (v));
        trace1 (gen,2)=sum (f1 (v))/length (f1 (v));
        trace2 (gen,1)=min (f2 (v));
        trace2 (gen,2)=sum (f2 (v))/length (f2 (v));
        trace3 (gen,1)=min (f1 (v)+f2 (v));
        trace3 (gen,2)=sum (f1 (v))/length (f1 (v))+sum (f2 (v))/length (f2 (v));
        gen=gen+1;
    end
    figure (1);clf;
    plot (trace1 (:,1));hold on; plot (trace1 (:,2),'-.');
    plot (trace1 (:,1),'.'); plot (trace1 (:,2),'.');grid;
    legend ('解的变化','种群均值的变化')
    xlabel ('迭代次数');ylabel ('目标函数值');
    figure (2);clf;
    plot (trace2 (:,1));hold on;
    plot (trace2 (:,2),'-.');
    plot (trace2 (:,1),'.');
    plot (trace2 (:,2),'.');grid;
    legend ('解的变化','种群均值的变化');
    xlabel ('迭代次数');ylabel ('目标函数值');
    figure (3);clf;
    plot (trace3 (:,1));hold on;
    plot (trace3 (:,2),'-.');
    plot (trace3 (:,1),'.');
    plot (trace3 (:,2),'.');grid;
    legend ('解的变化','种群均值的变化');
    xlabel ('迭代次数');ylabel ('目标函数值');
    figure (4);clf;plot (f1 (v));hold on;
    plot (f2 (v),'r-.');grid;
```

经过 50 次遗传迭代后的结果如图 7.50～图 7.53 所示。

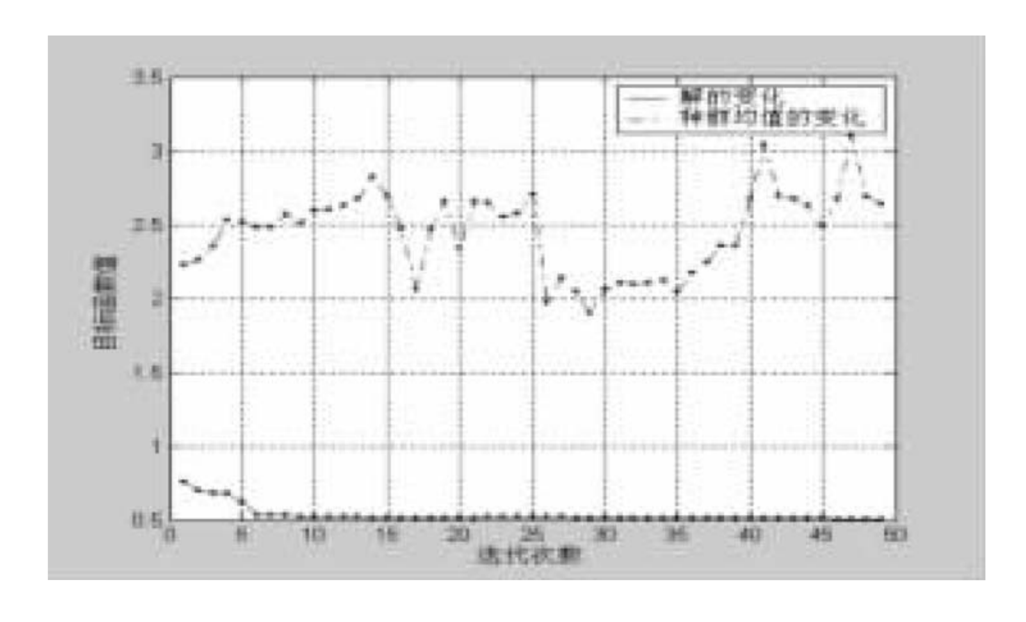



图 7.50　经过 50 次迭代后第一目标函数的最优解及性能跟踪

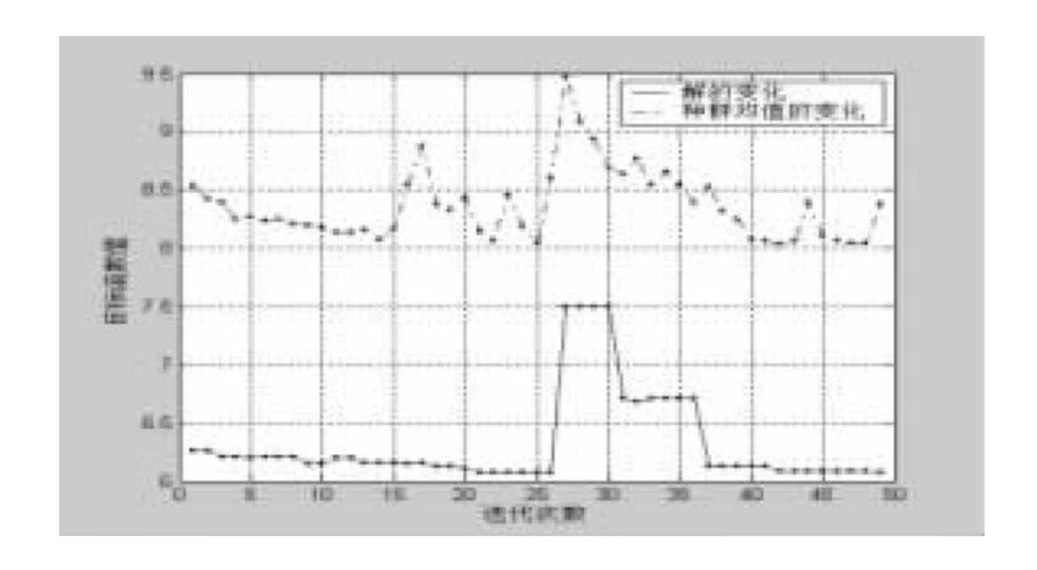



图 7.51　经过 50 次迭代后第二目标函数的最优解及性能跟踪

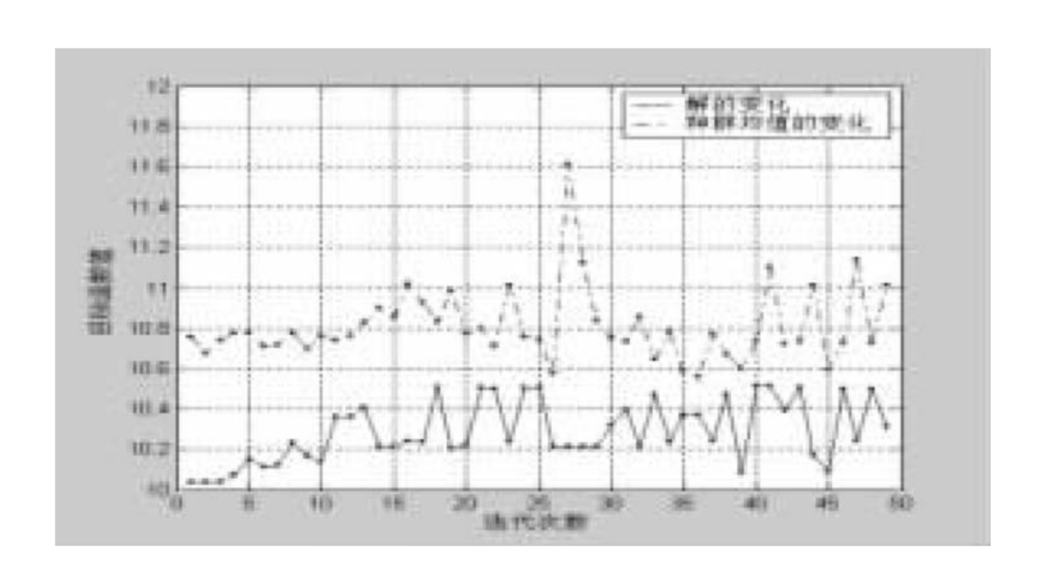



图 7.52　经过 50 次迭代后的两目标函数和的最优解及性能跟踪

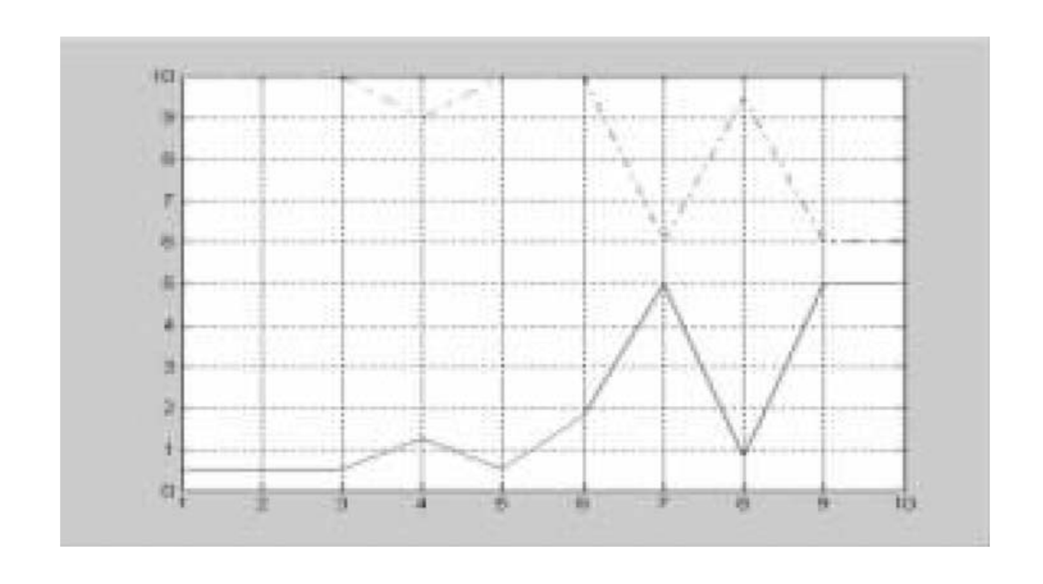


图 7.53　经过 50 次迭代后种群的第一目标函数值与第二目标函数值

# 第八章 使用 MATLAB 遗传算法工具

最新发布的 MATLAB 7.0 Release 14 包含一个专门设计的遗传算法与直接搜索工具箱(Genetic Algorithm and Direct Search Toolbox)。使用遗传算法与直接搜索工具箱，可以扩展 MATLAB 及其优化工具箱在处理优化问题方面的能力，可以处理传统的优化技术难以解决的问题，包括那些难以定义或不便于进行数学建模的问题，可以解决目标函数较复杂的问题，比如目标函数不连续或具有高度非线性、随机性以及目标函数不可微的情况。

本章 8.1 节首先综合介绍这个遗传算法与直接搜索工具箱，其余各节具体介绍该工具箱中的遗传算法工具及其使用方法。直接搜索工具箱将在第九章详细介绍。

## 8.1 遗传算法与直接搜索工具箱概述

本节介绍 MATLAB 的 GADS(遗传算法与直接搜索)工具箱的特点、图形用户界面及运行要求，解释如何编写待优化函数的 M 文件，且通过举例加以阐明。

### 8.1.1 工具箱的特点

GADS 工具箱是一系列函数的集合，它们扩展了优化工具箱和 MATLAB 数值计算环境的性能。遗传算法与直接搜索工具箱包含了要使用遗传算法和直接搜索算法来求解优化问题的一些例程。这些算法使我们能够求解那些标准优化工具箱范围之外的各种优化问题。所有工具箱函数都是 MATLAB 的 M 文件，这些文件由实现特定优化算法的 MATLAB 语句所写成。

使用语句

```
type function_name
```

就可以看到这些函数的 MATLAB 代码。我们也可以通过编写自己的 M 文件来实现和扩展遗传算法和直接搜索工具箱的性能，也可以将该工具箱与 MATLAB 的其他工具箱或 Simulink 结合使用来求解优化问题。

工具箱函数可以通过图形界面或 MATLAB 命令行来访问，它们是用 MATLAB 语言编写的，对用户开放，因此可以查看算法、修改源代码或生成用户函数。

遗传算法与直接搜索工具箱可以帮助我们求解那些不易用传统方法解决的问题，譬如表查找问题等。

遗传算法与直接搜索工具箱有一个精心设计的图形用户界面，可以帮助我们直观、方便、快速地求解最优化问题。

**1．功能特点**

遗传算法与直接搜索工具箱的功能特点如下：

（1）图形用户界面和命令行函数可用来快速地描述问题、设置算法选项以及监控进程。

（2）具有多个选项的遗传算法工具可用于问题创建、适应度计算、选择、交叉和变异。

（3）直接搜索工具实现了一种模式搜索方法，其选项可用于定义网格尺寸、表决方法和搜索方法。

（4）遗传算法与直接搜索工具箱函数可与 MATLAB 的优化工具箱或其他的 MATLAB 程序结合使用。

（5）支持自动的 M 代码生成。

**2．图形用户界面和命令行函数**

遗传算法工具函数可以通过命令行和图形用户界面来使用遗传算法。直接搜索工具函数也可以通过命令行和图形用户界面来进行访问。图形用户界面可用来快速地定义问题，设置算法选项，对优化问题进行详细定义。

遗传算法和直接搜索工具箱还同时提供了用于优化管理、性能监控及终止准则定义的工具，同时还提供了大量的标准算法选项。

在优化运行的过程中，可以通过修改选项来细化最优解，更新性能结果。用户也可以提供自己的算法选项来定制工具箱。

**3．使用其他函数和求解器**

遗传算法与直接搜索工具箱和 MATLAB 及优化工具箱是紧密结合在一起的。用户可以用遗传算法或直接搜索算法来寻找最佳起始点，然后利用优化工具箱或用 MATLAB 程序来进一步寻找最优解。通过结合不同的算法，可以充分地发挥 MATLAB 和工具箱的功能以提高求解的质量。对于某些特定问题，使用这种方法还可以得到全局(最优)解。

**4．显示、监控和输出结果**

遗传算法与直接搜索工具箱还包括一系列绘图函数，用来可视化优化结果。这些可视化功能直观地显示了优化的过程，并且允许在执行过程中进行修改。

该工具箱还提供了一些特殊绘图函数，它们不仅适用于遗传算法，还适用于直接搜索算法。适用于遗传算法的函数包括函数值、适应度值和函数估计。适用于直接搜索算法的函数包括函数值、分值直方图、系谱、适应度值、网格尺寸和函数估计。这些函数可以将多个绘图一并显示，可直观方便地选取最优曲线。另外，用户也可以添加自己的绘图函数。

使用输出函数可以将结果写入文件，产生用户自己的终止准则，也可以写入用户自己的图形界面来运行工具箱求解器。除此之外，还可以将问题的算法选项导出，以便日后再将它们导入到图形界面中去。

**5．所需的产品支持**

遗传算法与直接搜索工具箱作为其他优化方法的补充，可以用来寻找最佳起始点，然后再通过使用传统的优化技术来进一步寻找最优解。

工具箱需要如下产品支持：

（1）MATLAB。

（2）优化工具箱。

**6. 相关产品**

与遗传算法与直接搜索工具箱相关的产品有：

(1) 统计工具箱——应用统计算法和概率模式。

(2) 神经网络工具箱——设计和仿真神经网络。

(3) 模糊逻辑工具箱——设计和仿真基于模糊逻辑的系统。

(4) 金融工具箱——分析金融数据和开发金融算法。

**7. 所需的系统及平台**

遗传算法和直接搜索工具箱对于运行环境、支持平台和系统的需求，可随时通过访问网站 http://www.mathworks.com/products/gads 了解最新发布的信息。

这里介绍的 MATLAB 7.0 Release 14 所需的最低配置是：Windows 系列操作系统，Pentium Ⅲ 500 MHz CPU，64 MB RAM，空闲硬盘空间 600 MB 以上。

## 8.1.2 编写待优化函数的 M 文件

为了使用遗传算法和直接搜索工具箱，首先必须编写一个 M 文件，来确定想要优化的函数。这个 M 文件应该接受一个行向量，并且返回一个标量。行向量的长度就是目标函数中独立变量的个数。本节将通过实例解释如何编写这种 M 文件。

**1. 编写 M 文件举例**

下面的例子展示了如何为一个想要优化的函数编写 M 文件。假定我们想要计算下面函数的最小值：

$$f(x_1, x_2) = x_1^2 - 2x_1x_2 + 6x_1 + x_2^2 - 6x_2$$

M 文件确定这个函数必须接受一个长度为 2 的行向量 $X$，分别与变量 $x_1$ 和 $x_2$ 相对应，并且返回一个标量 $Z$，其值等于该函数的值。为了编写这个 M 文件，执行如下步骤：

(1) 在 MATLAB 的 File 菜单中选择 New 菜单项。

(2) 选择 M-File，在编辑器中打开一个新的 M 文件。

(3) 在该 M 文件中，输入下面两行代码：

```
function z=my_fun(x)
z=x(1)^2-2*x(1)*x(2)+6*x(1)+x(2)^2-6*x(2);
```

(4) 在 MATLAB 路径指定的目录中保存该 M 文件。

为了查看该 M 文件是否返回正确的值，可键入

```
my_fun([2 3])
ans=
     -5
```

注意：在运行遗传算法工具或模式搜索工具时，不要使用编辑器或调试器来调试目标函数的 M 文件，否则会导致在命令窗口出现 Java 异常消息，并且使调试更加困难。

**2. 最大化与最小化**

遗传算法和直接搜索工具箱中的优化函数总是使目标函数或适应度函数最小化，也就是说，它们求解如下形式的问题：

$$\min_x f(x)$$

如果想要求出函数 $f(x)$ 的最大值，可以转而求取函数 $g(x) = -f(x)$ 的最小值，因为

函数 $g(x)$ 最小值出现的地方与函数 $f(x)$ 最大值出现的地方相同。

例如想要求前面所描述的函数 $f(x_1, x_2)=x_1^2-2x_1x_2+6x_1+x_2^2-6x_2$ 的最大值，这时，我们应当编写一个 M 文件来计算，求函数 $g(x)=-f(x_1, x_2)=-(x_1^2-2x_1x_2+6x_1+x_2^2-6x_2)$ 的最小值。

**3.** 自动代码生成

遗传算法与直接搜索工具箱提供了自动代码生成特性，可以自动生成求解优化问题所需要的 M 文件。例如，图 8.1 所示就是使用遗传算法工具的自动代码生成特性所产生的 M 文件。

另外，图形用户界面所输出的优化结果可以作为对来自命令行调用代码的一种解释，这些代码还用于使例程和保护工作自动化。

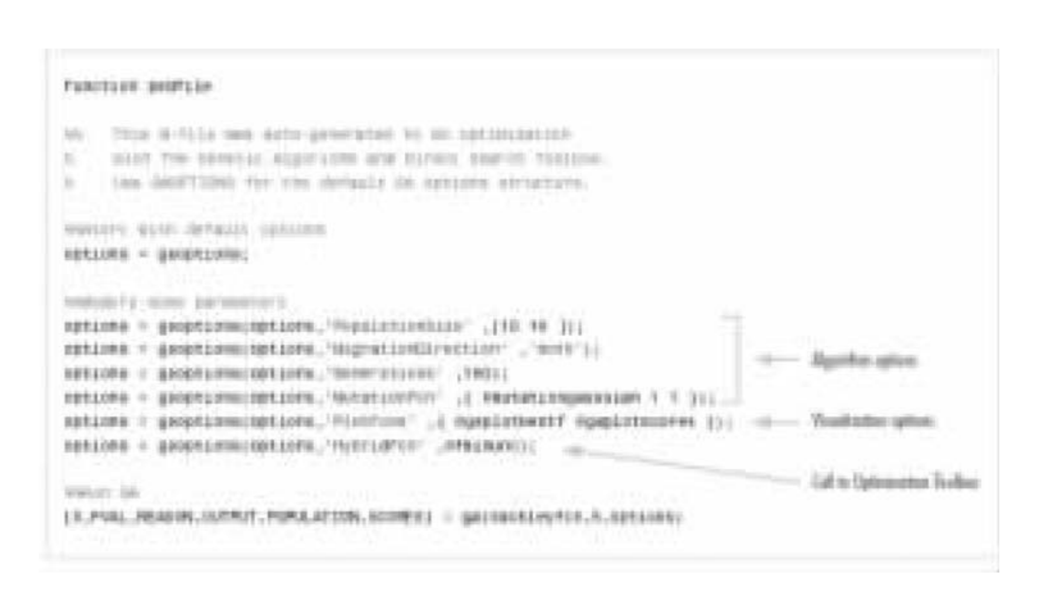


图 8.1　遗传算法 M 文件代码的自动生成

# 8.2　使用遗传算法工具初步

遗传算法与直接搜索工具箱包含遗传算法工具和直接搜索工具。从本节至章末，将主要介绍其中的遗传算法工具及其使用方法。

本节主要介绍遗传算法工具使用的初步知识，内容包括：遗传算法使用规则，遗传算法工具的使用方式，举例说明如何使用遗传算法来求解一个优化问题，遗传算法的一些基本术语，遗传算法的工作原理与工作过程。

## 8.2.1　遗传算法使用规则

遗传算法是一种基于自然选择、生物进化过程来求解问题的方法。遗传算法反复修改对于个体解决方案的种群。在每一步中，遗传算法随机地从当前种群中选择若干个体作为父辈，并且使用它们产生下一代的子种群。在连续若干代之后，种群朝着优化解的方向进化。我们可以用遗传算法来求解各种不适宜于用标准优化算法求解的优化问题，包括目标函数不连续、不可微、随机或高度非线性的问题。

遗传算法在每一步使用下列三类规则从当前种群来创建下一代：

(1) 选择规则(Selection Rules)：选择对下一代种群有贡献的个体(称为父辈)。

(2) 交叉规则(Crossover Rules):将两个父辈结合起来构成下一代的子辈种群。

(3) 变异规则(Mutation Rules):施加随机变化给父辈个体来构成子辈。

遗传算法与标准优化算法主要在两个方面有所不同,它们的比较情况归纳于表8.1中。

表 **8.1** 遗传算法与标准优化算法比较

| 标准算法 | 遗传算法 |
| --- | --- |
| 每次迭代产生一个单点,点的序列逼近一个优化解 | 每次迭代产生一个种群,种群逼近一个优化解 |
| 通过确定性的计算在该序列中选择下一个点 | 通过随机进化选择计算来选择下一代种群 |

### 8.2.2 遗传算法使用方式

遗传算法工具有两种使用方式:

(1) 以命令行方式调用遗传算法函数 ga。

(2) 通过图形用户界面使用遗传算法工具。

本节对这两种方式做一个简要的介绍。

**1. 在命令行调用函数 ga**

对于在命令行使用遗传算法,可以用下列语法调用遗传算法函数 ga:

```
[x fval]=ga(@fitnessfun, nvars, options)
```

其中,@fitnessfun 是适应度函数句柄;nvars 是适应度函数的独立变量的个数;options 是一个包含遗传算法选项参数的结构。如果不传递选项参数,则 ga 使用它本身的缺省选项值。

函数所给出的结果为:fval——适应度函数的最终值;x——最终值到达的点。

我们可以十分方便地把遗传算法工具输出的结果直接返回到 MATLAB 的 workspace(工作空间),或以不同的选项从 M 文件多次调用函数 ga 来运行遗传算法。

调用函数 ga 时,需要提供一个选项结构 options。后面的有关章节对于在命令行中使用函数 ga 和创建选项结构 options 提供了详细的描述。

**2. 通过 GUI 使用遗传算法**

遗传算法工具有一个图形用户界面 GUI,它使我们可以使用遗传算法而不用工作在命令行方式。打开遗传算法工具,可键入以下命令:

```
gatool
```

打开的遗传算法工具图形用户界面如图 8.2 所示。

使用遗传算法工具首先必须输入下列信息:

(1) Fitness function(适应度函数)——欲求最小值的目标函数。输入适应度函数的形式为@fitnessfun,其中 fitnessfun.m 是计算适应度函数的 M 文件。在 8.1.2 节"编写待优化函数的 M 文件"里已经解释了如何编写这种 M 文件。符号@产生一个对于函数 fitnessfun 的函数句柄。

(2) Number of variables(变量个数)——适应度函数输入向量的长度。对于"编写待优化函数的 M 文件"一节所描述的函数 my_fun,它的参数是 2。




图 8.2　遗传算法工具

单击“Start”按钮，运行遗传算法，将在“Status and results（状态与结果）”窗格中显示出相应的运行结果。

在“Options”窗格中可以改变遗传算法的选项。为了查看窗格中所列出的各类选项，可单击与之相连的符号“+”。

### 8.2.3　举例：Rastrigin 函数

本节通过一个例子讲述如何寻找 Rastrigin 函数的最小值和显示绘制的图形。Rastrigin 函数是最常用来测试遗传算法的一个典型函数。Rastrigin 函数的可视化图形显示具有多个局部最小值和一个全局最小值，遗传算法可帮助我们确定这种具有多个局部最小值函数的最优解。

**1. Rastrigin** 函数

具有两个独立变量的 Rastrigin 函数定义为

$$Ras(x) = 20 + x_1^2 + x_2^2 - 10(\cos 2\pi x_1 + \cos 2\pi x_2)$$

Rastrigin 函数的图形如图 8.3 所示。

工具箱包含一个 M 文件，即 rastriginsfcn.m，是用来计算 Rastrigin 函数值的。

如图 8.3 所示，Rastrigin 函数有许多局部最小值——在图上显示为“谷底（Valleys）”。然而，该函数只有一个全局最小值，出现在 $x-y$ 平面上的点[0，0]处，正如图中竖直线指示的那样，函数的值在那里是 0。在任何不同于[0，0]的局部最小点处，Rastrigin 函数的值均大于 0。局部最小处距原点越远，该点处 Rastrigin 函数的值越大。

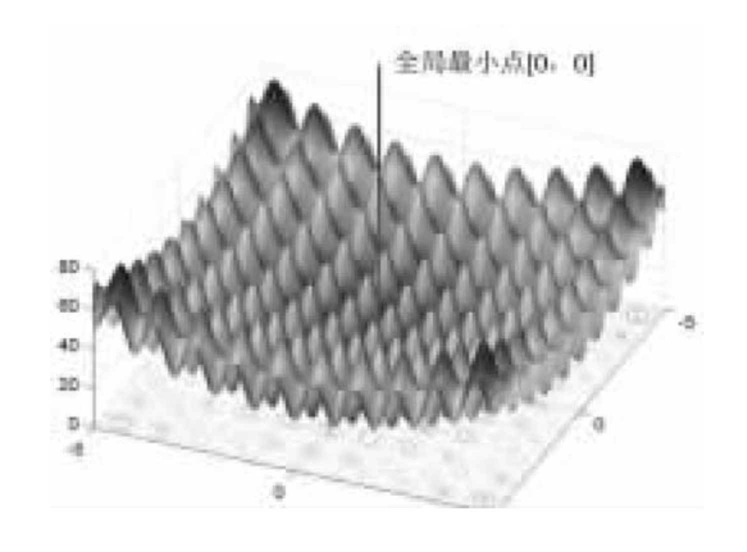



图 8.3　Rastrigin 函数图形

Rastrigin 函数之所以最常用来测试遗传算法，是因为它有许多局部最小点，使得用标准的、基于梯度的查找全局最小的方法十分困难。

图 8.4 所示是 Rastrigin 函数的轮廓线，它显示出最大、最小交替变化的情形。

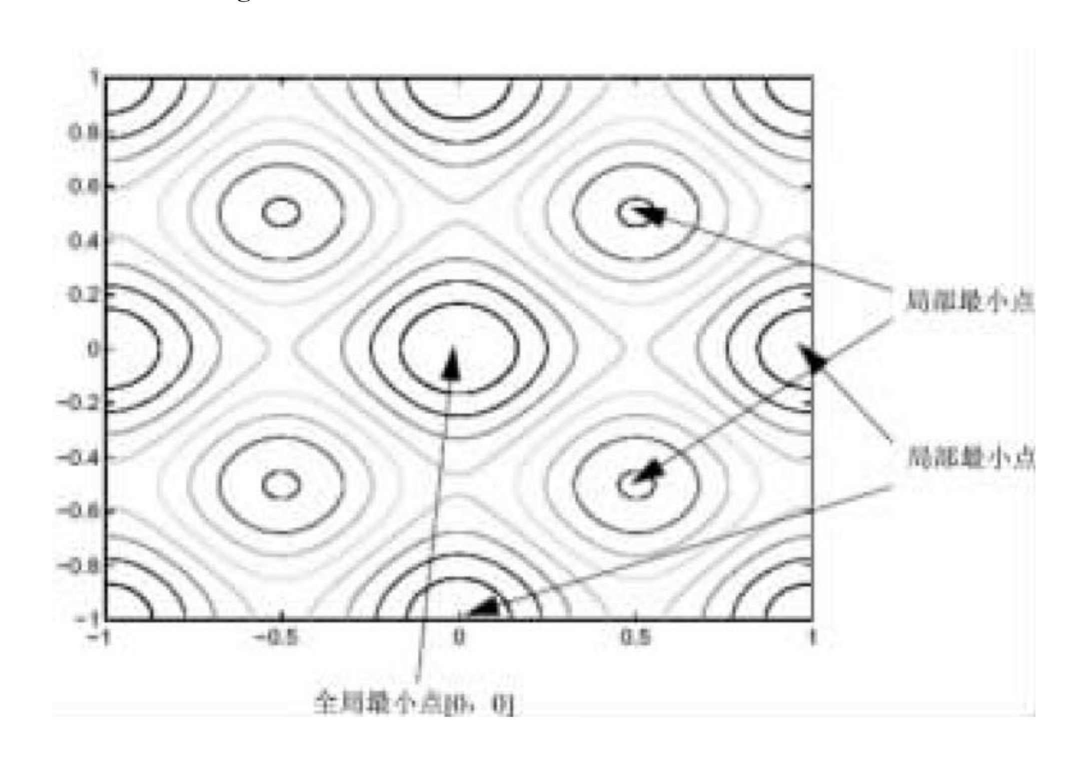



图 8.4　Rastrigin 函数的轮廓线

**2．寻找 Rastrigin 函数的最小值**

在使用遗传算法时，因为是使用随机数据来进行搜索，所以该算法每一次运行时所返回的结果会稍微有些不同。

为了查找最小值，执行下列步骤：

（1）在命令行键入 gatool，打开遗传算法工具。

（2）在遗传算法工具的相应栏目中输入适应度函数和变量个数。在“Fitness function（适应度函数）”文本框中，输入“@rastriginsfcn”；在“Number of variables（变量个数）”文本框中，输入“2”，这就是 Rastrigin 函数独立变量的个数，如图 8.5 所示。

（3）在“Run solver（运行求解器）”窗格中单击“Start”按钮，如图 8.6 所示。




图 8.5　输入适应度函数与变量个数




图 8.6　单击“Run solver(运行求解器)”的“Start”按钮

在算法运行的同时，“Current generation(当前代数)”文本框中显示出当前的代数。通过单击“Pause(暂停)”按钮，可以使算法临时暂停一下。当这样做的时候，该按钮的名字变为“Resume(恢复)”。为了从暂停处恢复算法的运行，可单击“Resume(恢复)”按钮。

当算法完成时，“Status and results”窗格出现如图 8.7 所示的情形。

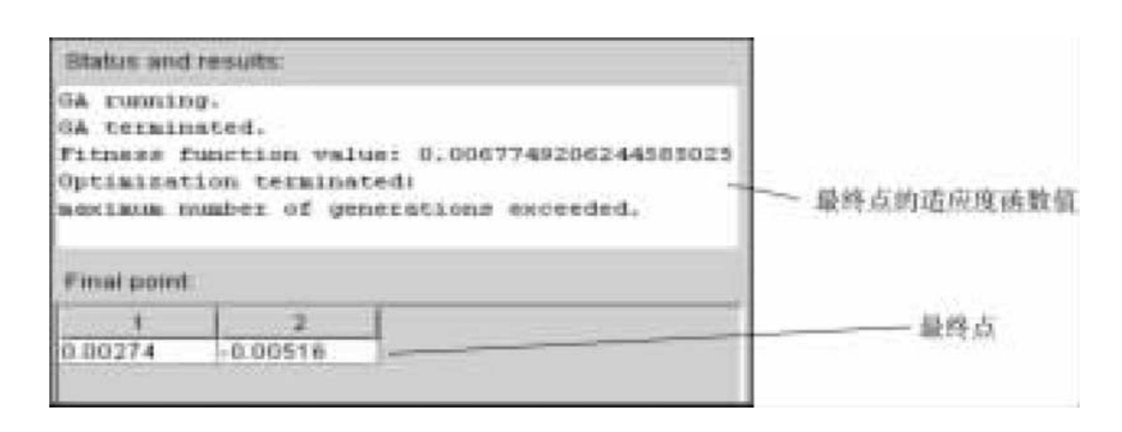



图 8.7　状态与结果显示

“Status and results”窗格中将显示下列信息：

(1) 算法终止时适应度函数的最终值：

　　Fitness function value：0.0067749206244585025

注意：所显示的值非常接近于 Rastrigin 函数的实际最小值 0。8.3.3 节“遗传算法举例”中描述了一些方法，可以用来得到更接近实际最小值的结果。

(2) 算法终止的原因：

　　Optimization terminated：

　　maximum number of generations exceeded.

即退出的原因是：超过最大代数而导致优化终止。

在本例中，算法在 100 代后结束，这是“Generations(代数)”选项的缺省值，此选项规定了算法计算的最大代数。

(3) 最终点。在本例中是[0.00274　−0.00516]。

**3.** 从命令行查找最小值

为了从命令行查找 Rastrigin 函数的最小值，可键入

```
[x fval reason]=ga(@rastriginsfcn, 2)
```
这将返回

```
x=
    0.0027 -0.0052
fval=
    0.0068
reason=
    Optimization terminated:
    maximum number of generations exceeded.
```

其中，x 是算法返回的最终点；fval 是该最终点处适应度函数的值；reason 是算法结束的原因。

**4.** 显示绘制图形

“Plots(绘图)”窗格可以显示遗传算法运行时所提供的有关信息的各种图形。这些信息可以帮助我们改变算法的选项，改进算法的性能。例如，为了绘制每一代适应度函数的最佳值和平均值，应选中复选框“Best fitness(最佳适应度)”，如图 8.8 所示。




图 8.8 “Plot(绘图)”对话框

当单击“Start”按钮时，遗传算法工具显示每一代适应度函数的最佳值和平均值的绘制图形。当算法停止时，所出现的图形如图 8.9 所示。

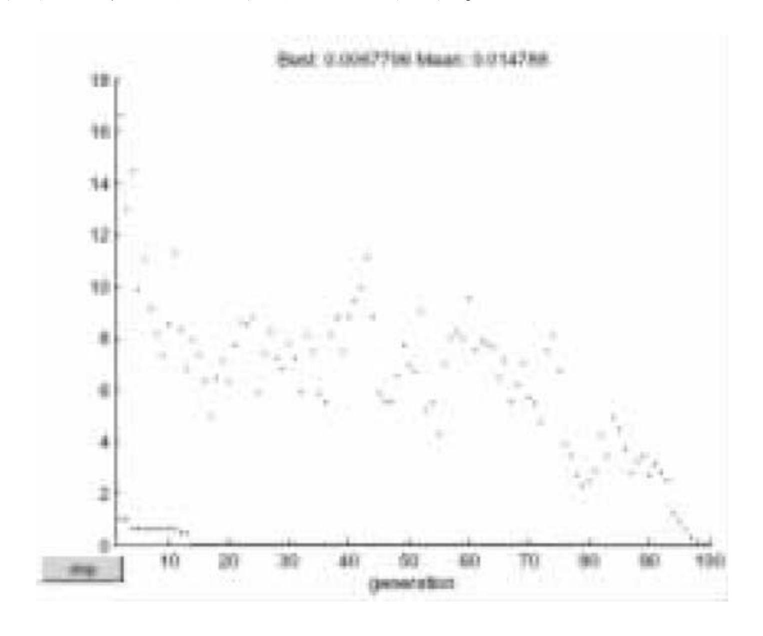



图 8.9 各代适应度函数的最佳值和平均值

在每一代中，图的底部的点表示最佳适应度值，而其上的点表示平均适应度值。图的

顶部还显示出当前代的最佳值和平均值。

为了得到最佳适应度值减少到多少为更好的直观图形，我们可以将图中 Y 轴的刻度改变为对数刻度。为此，需进行如下操作：

（1）从“Plot(绘图)”窗格的“Edit(菜单)”菜单中选择“Axes Properties(坐标轴属性)”，打开属性编辑器，如图 8.10 所示。

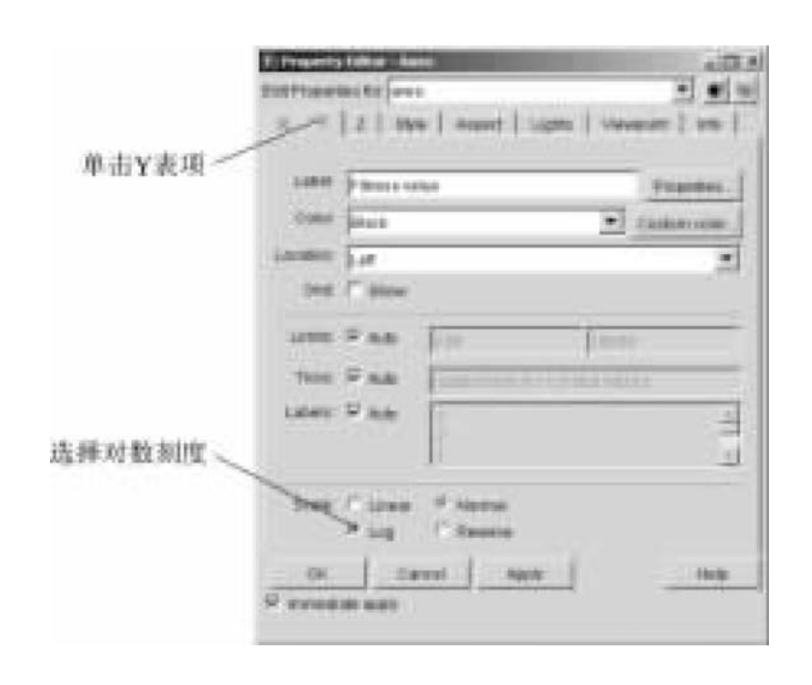



图 8.10　绘图属性编辑器

（2）单击 Y 表项。

（3）在“Scale(刻度)”窗格中选择“Log(对数)”。

绘制的图形如图 8.11 所示。

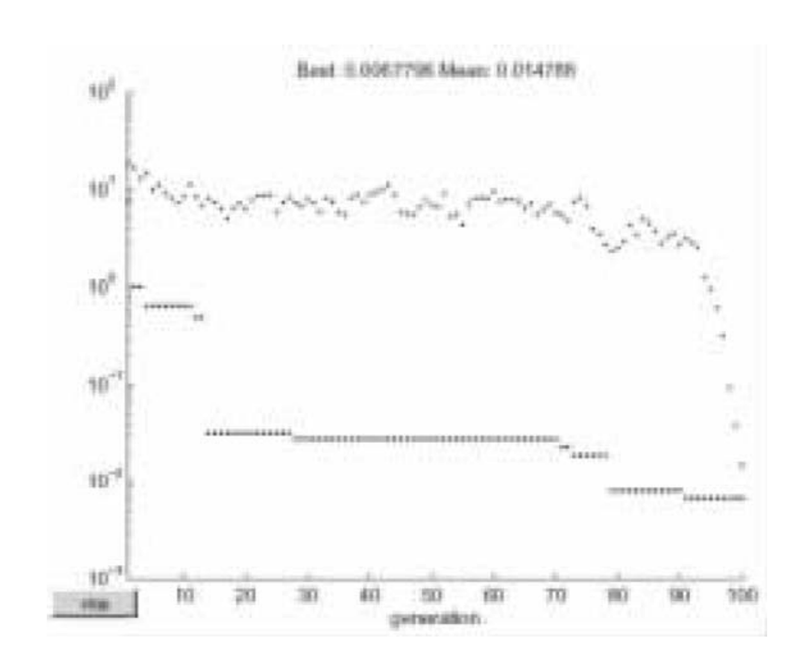



图 8.11　每一代适应度函数最佳值和平均值的对数图形

典型情况下，在早期各代中，当个体离理想值较远时，最佳值会迅速得到改进。在后来各代中，种群越接近最佳点，最佳值改进得越慢。

## 8.2.4 遗传算法的一些术语

本节解释遗传算法的一些基本术语，主要包括：

(1) 适应度函数(Fitness Functions)。

(2) 个体(Individuals)。

(3) 种群(Populations)和代(Generations)。

(4) 多样性或差异(Diversity)。

(5) 适应度值(Fitness Values)和最佳适应度值(Best Fitness Values)。

(6) 父辈和子辈(Parents and Children)。

**1. 适应度函数**

所谓适应度函数，就是想要优化的函数。对于标准优化算法而言，这个函数称为目标函数。该工具箱总是试图寻找适应度函数的最小值。

我们可以将适应度函数编写为一个 M 文件，作为输入参数传递给遗传算法函数。

**2. 个体**

一个个体是可以施加适应度函数的任意一点。一个个体的适应度函数值就是它的得分或评价。例如，如果适应度函数是

$$f(x_1, x_2, x_3) = (2x_1 + 1)^2 + (3x_2 + 4)^2 + (x_3 - 2)^2$$

则向量(2，−3，1)就是一个个体，向量的长度就是问题中变量的个数。个体(2，−3，1)的得分是 $f(2, -3, 1) = 51$。

个体有时又称为基因组或染色体组(Genome)，个体的向量项称为基因(Genes)。

**3. 种群与代**

所谓种群，是指由个体组成的一个数组或矩阵。例如，如果个体的长度是 100，适应度函数中变量的个数为 3，我们就可以将这个种群表示为一个 100×3 的矩阵。相同的个体在种群中可以出现不止一次。例如，个体(2，−3，1)就可以在数组的行中出现多次。

每一次迭代，遗传算法都对当前种群执行一系列的计算，产生一个新的种群。每一个后继的种群称为新的一代。

**4. 多样性**

多样性或差异(Diversity)涉及一个种群的各个个体之间的平均距离。若平均距离大，则种群具有高的多样性；否则，其多样性低。在图 8.12 中，左边的种群具有高的多样性，亦即差异大；而右边的种群多样性低，亦即差异小。

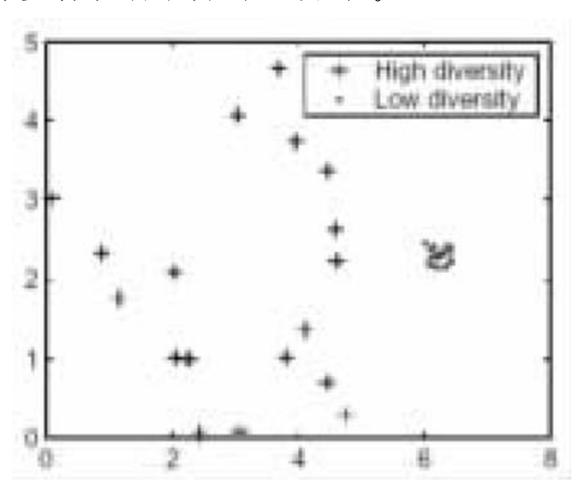



图 8.12 种群多样性比较

多样性是遗传算法必不可少的本质属性，这是因为它能使遗传算法搜索一个比较大的解的空间区域。

**5．**适应度值和最佳适应度值

个体的适应度值就是该个体的适应度函数的值。由于该工具箱总是查找适应度函数的最小值，因此一个种群的最佳适应度值就是该种群中任何个体的最小适应度值。

**6．**父辈和子辈

为了生成下一代，遗传算法在当前种群中选择某些个体(称为父辈)，并且使用它们来生成下一代中的个体(称为子辈)。典型情况下，该算法更可能选择那些具有较佳适应度值的父辈。

### 8.2.5 遗传算法如何工作

本节简要介绍遗传算法的工作原理或工作过程，内容包括算法要点、初始种群、生成下一代、后一代的绘图和算法的停止条件。

**1．**算法要点

下面的要点总结了遗传算法是如何工作的：

(1) 算法创建一个随机种群。

(2) 算法生成一个新的种群序列，即新的一代。在每一步，该算法都使用当前一代中的个体来生成下一代。为了生成新一代，算法执行下列步骤：

(a) 通过计算其适应度值，给当前种群的每一个成员打分。

(b) 确定原来的适应度值的比例尺度，将其转换为更便于使用的范围内的值。

(c) 根据它们的适应度选择父辈。

(d) 由父辈产生子辈。子辈的产生可以通过随机改变一个单个父辈，亦即变异(Mutation)来进行，也可以通过组合一对父辈的向量项，亦即交叉(Crossover)来进行。

(e) 用子辈替换当前种群，形成下一代。

(3) 若停止准则之一得到满足，则该算法停止(关于停止准则，可参见 8.2.5 节“遗传算法如何工作”中的“算法的停止条件”)。

**2．**初始种群

遗传算法总是以产生一个随机的初始种群开始，如图 8.13 所示。

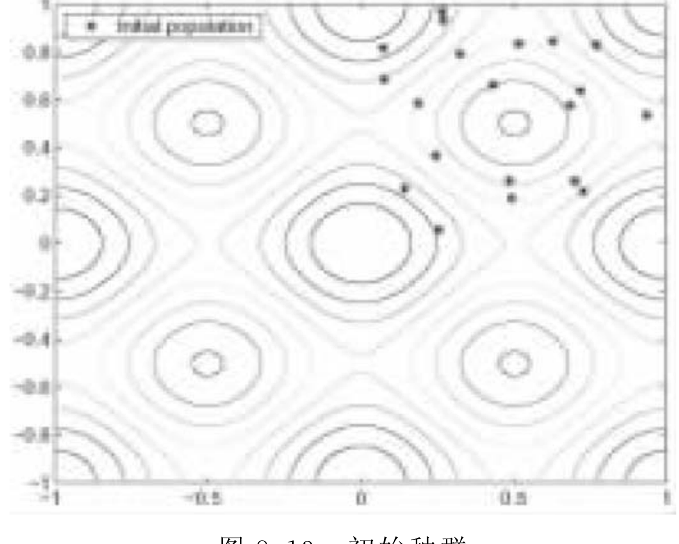



图 8.13　初始种群

在本例中，初始种群包含20个个体，这恰好是“Population(种群)”选项中的“Population size(种群尺度)”的缺省值。

注意：初始种群中的所有个体均处于图上右上角的那个象限，也就是说，它们的坐标处于0和1之间，这是因为“Population”选项中的“Initial range(初始范围)”的缺省值是[0；1]。

如果已知函数的最小点大约位于何处，就可以设置一个适当的“Initial range”，以便使该点处于那个范围的中间附近。例如，如果确信Rastrigin函数的最小值在点[0，0]附近，那么就可以直接设置“Initial range”为[－1；1]。然而，正如本例所显示的那样，即使没有给“Initial range”设置一个理想的值，遗传算法也还是能够找到那个最小值。

**3.** 产生下一代

在每一步中，遗传算法使用当前种群来产生子辈，即获得下一代。算法在当前种群中选择一组个体，称为父辈，这些父辈将其Genes(亦即其向量中的项)贡献给它们的子辈。遗传算法通常选择那些具有较好适应度值的个体作为父辈。我们可以在“Selection(选择)”选项的“Selection function(选择函数)”文本框中指定遗传算法用来选择父辈的函数。

遗传算法对于下一代产生三类子辈：优良子辈、交叉子辈和变异子辈。

(1) 优良子辈(Elite children)是在当前代中具有最佳适应度值的那些个体。这些个体子辈存活到下一代。

(2) 交叉子辈(Crossover children)是由一对父辈向量组合产生的。

(3) 变异子辈(Mutation children)是对一个单个父辈引入随机改变即变异产生的。

图8.14表示了这三个类型的子辈。

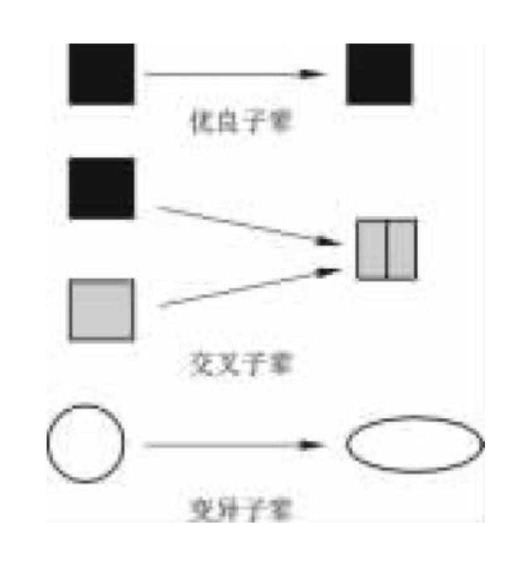



图8.14　三类子辈

在8.3.3节“遗传算法举例”的“变异与交叉”部分中将解释如何指定遗传算法产生的每一类子辈的数目，以及用来执行完成交叉和变异的函数。

**4.** 交叉子辈

算法通过组合当前种群中的父辈对(Pair)来产生交叉子辈。在子辈向量的每一个相同位置处，缺省的交叉函数在两个父辈之一的相同位置处随机选择一项(即基因)，并将它指派给其子辈。

**5.** 变异子辈

算法通过随机改变个体父辈中的基因而产生变异子辈。按照缺省，算法给父辈增加一

个高斯分布的随机向量。

图 8.15 所示为初始种群的子辈，也即第二代种群，并且指出它们是否为优良子辈、交叉子辈或变异子辈。

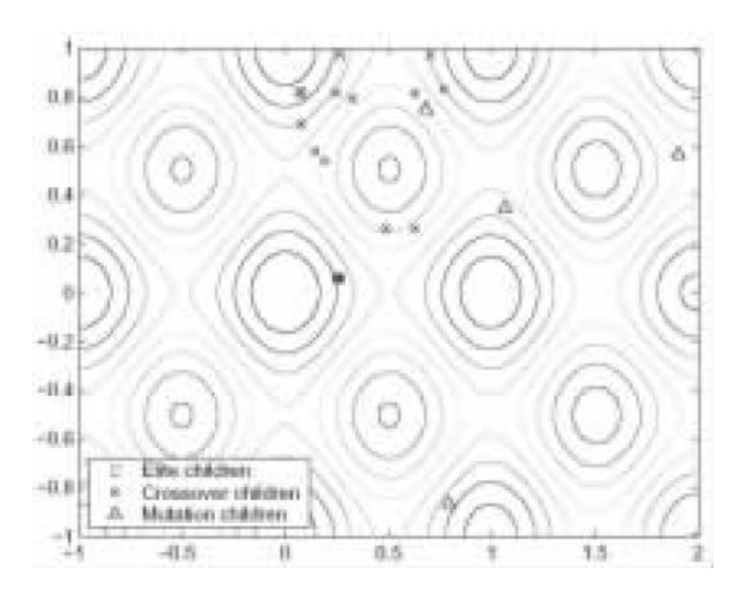



图 8.15　初始种群的子辈

**6.** 后代图形绘制

图 8.16 所示为在迭代 60、80、95、100 次时种群的图形。

图 8.16　在迭代 60、80、95、100 次时的种群

随着代数的增加，种群中的个体靠近在一起，且逼近最小值点[0，0]。

**7.** 算法的停止条件

遗传算法使用下列5个条件来确定何时停止：

(1) Generations(代数)——当产生的代的数目达到规定的代数的值时，算法停止。

(2) Time limit(时限)——在运行时间的秒数等于时限时，算法停止。

(3) Fitness limit(适应度限)——当适应度函数的值对于当前种群的最佳点小于或等于适应度限时，算法停止。

(4) Stall generations(停滞代数)——在连续繁殖的时间序列中，若长时间不繁殖新代，亦即目标函数无改进，到达停滞代数规定的代数时，则算法停止。

(5) Stall time limit(停滞时限)——在秒数等于停滞时限的时间间隔期间，若目标函数无改进，则算法停止。

若这5个条件中任何一个条件一旦满足，则该算法停止。我们可以在遗传算法工具的"Stopping criteria(停止标准)"选项中指定这些标准的值。它们的缺省值如图8.17所示。

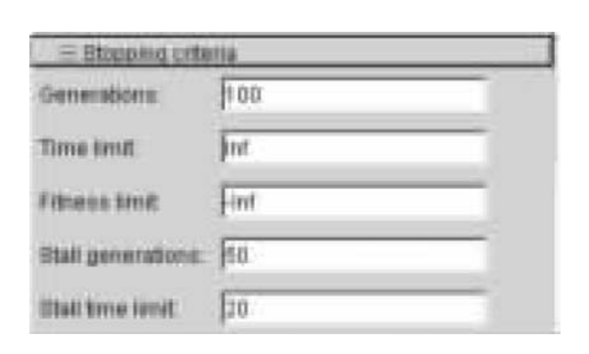



图8.17　停止标准的缺省值

当运行遗传算法时，"Status(状态)"面板显示这些导致算法停止的标准。

"Time limit(时限)"选项与"Stall time limit"选项可以用来防止算法运行过长的时间。如果算法由于这两个条件之一而停止，则可以通过相应增加"Time limit"或"Stall time limit"的值来改善运行的结果。

# 8.3　使用遗传算法工具求解问题

本节首先概括介绍使用遗传算法工具GUI的一般步骤，然后介绍如何从命令行使用遗传算法工具，最后通过举例详细说明如何使用遗传算法工具来求解优化问题。

## 8.3.1　使用遗传算法工具GUI

前面已经介绍了使用遗传算法工具的初步知识。本节将简要归纳使用遗传算法工具GUI来求解优化问题的一般步骤，内容包括：打开遗传算法工具；在遗传算法工具中定义问题；运行遗传算法；暂停和停止运算；图形显示；创建用户图形函数；复现运行结果；设置选项参数；输入输出参数及问题；从最后种群继续运行遗传算法。

**1.** 打开遗传算法工具

在MATLAB窗口中输入gatool，打开、进入遗传算法工具。初启时的界面如图8.18所示。



图 8.18　遗传算法工具初启时的界面

**2**．在遗传算法工具中定义问题

在下列两个文本框中定义所要解决的问题：

(1) 适应度函数——求解的问题是求目标函数的最小值。输入一个计算适应度函数的 M 文件函数的句柄。

(2) 变量个数——适应度函数的独立变量个数。

注意：当运行遗传算法工具时不要用“Editor/Debugger(编辑/调试)”功能来调试目标函数的 M 文件，而要从命令行直接调用目标函数或把 M 文件输入到遗传算法函数 ga。为了方便调试，可以在遗传算法工具中把问题输出到 MATLAB 工作窗口中。

如图 8.19 所示，输入前面章节所介绍的 Rastrigin 函数或 my_fun 函数作为适应度函数，它们的变量个数为 2。




图 8.19　输入适应度函数与变量个数

**3**．运行遗传算法

要运行遗传算法，在“Run solver(运行求解器)”中单击“Start”按钮，如图 8.20 所示。

这时，在“Current generation(当前代)”文本框中显示当前代的数目，在“Status and results”窗格中显示“GA running.”等信息，如图 8.21 所示。

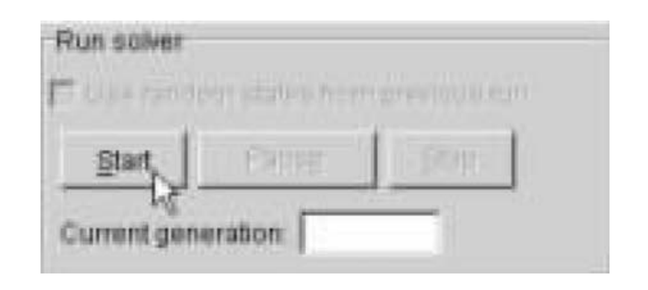



图 8.20　单击“Start”按钮

Current generation: 48
Status and results:
GA running.

图 8.21　当前代数和状态与结果窗格

当遗传算法停止时，“Status and results”窗格显示：

（1）“GA terminated(GA 终止)”信息。

（2）最后一代最佳个体的适应度函数值。

（3）算法停止的原因。

（4）最终点的坐标。

图 8.22 中显示了当运行例子“Rastrigin 函数”遗传算法停止时的信息。

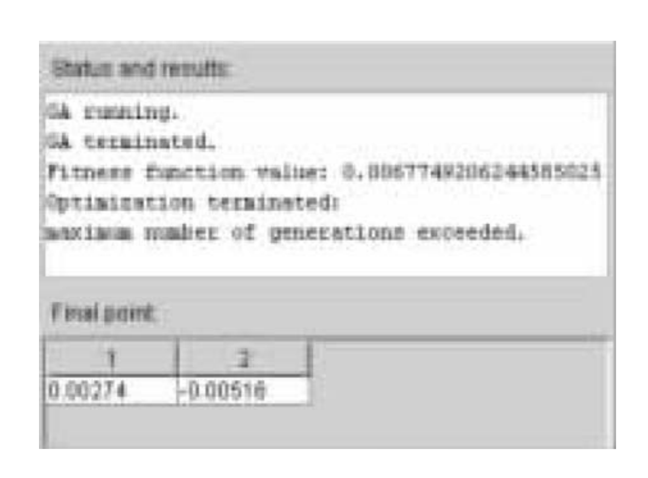



图 8.22　Rastrigin 函数的遗传算法运行结果

在遗传算法工具中，当遗传算法运行时可以更改多个参数设置，所做的改变将被应用到下一代，即在下一代将按照新设置的参数运行。在下一代开始但尚未应用改变的参数之前，在“Status and results”窗格中显示信息“Changes pending.”。而在下一代开始且应用了改变的参数时，在“Status and results”窗格中显示信息“Changes applied.”。这样，在遗传算法运行时更改了参数设置后产生的输出信息如图 8.23 所示。

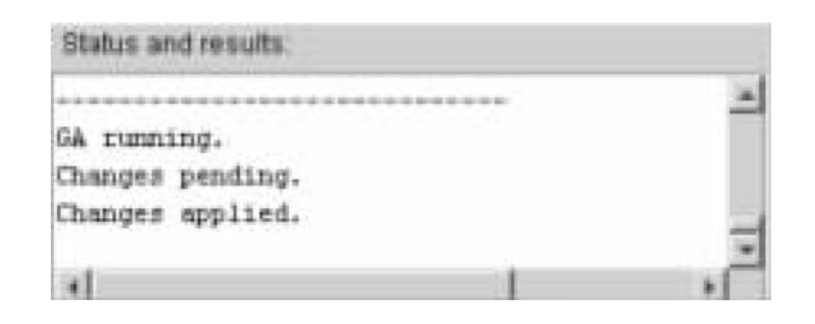



图 8.23　遗传算法运行时更改了参数设置

**4．**暂停和停止运算

遗传算法的暂停和停止运行，可以通过下面的操作继续运行：

（1）单击“Pause(暂停)”按钮，算法暂停运行。该按钮上的文字变为“Resume(恢复)”。单击“Resume” 按钮，即恢复遗传算法继续运行。

（2）单击“Stop(停止)”按钮，算法停止运行。“Status and results”窗口中将显示停止运行时当前代最佳点的适应度函数值。

注意：如果单击“Stop”按钮，然后通过单击“Start”按钮再次运行时，遗传算法将以新的随机初始种群或在“Initial population(初始种群)”文本框中专门指定的种群运行。如果需要在算法停止后能再次恢复运行，则可以通过交替地单击“Pause”和“Resume”按钮来控制算法暂停或继续运行。

遗传算法的停止运行常常是通过设置算法停止准则来进行控制的。使用停止准则，设置停止准则参数，可以解决遗传算法在何时停止运行的控制问题。这样，也就不用通过单击“Stop” 按钮来人为地控制算法运行的停止。遗传算法有五个停止准则或条件，其中任何一个条件满足，算法即停止运行。这些停止准则是：

（1）代数——算法运行到规定的代数。

（2）时限——算法运行到规定的时间。

（3）适应度限——当前代的最佳适应度值小于或等于规定的值。

（4）停滞代数——适应度函数值在运行规定的代数后没有改进。

（5）停滞时限——适应度函数值在运行规定时间后没有改进。

如果想使算法一直运行到单击“Pause”或“Stop”按钮时才停下来，可以改变这些停止准则的参数值：

（1）设置“Generations(代数)”为 Inf。

（2）设置“Time”为 Inf。

（3）设置“Fitness limit”为－Inf。

（4）设置“Stall generations”为 Inf。

（5）设置“Stall time limit”为 Inf。

图 8.24 显示了这些更改后的设置。

注意：在命令行中调用遗传算法函数 ga 时，并不使用这些参数设置，就好像是不按下“Ctrl＋C”键，函数就会永远运行而不会停止。其实相反，可以设置“Generations”或者“Time”做为限值来控制算法停止运行。

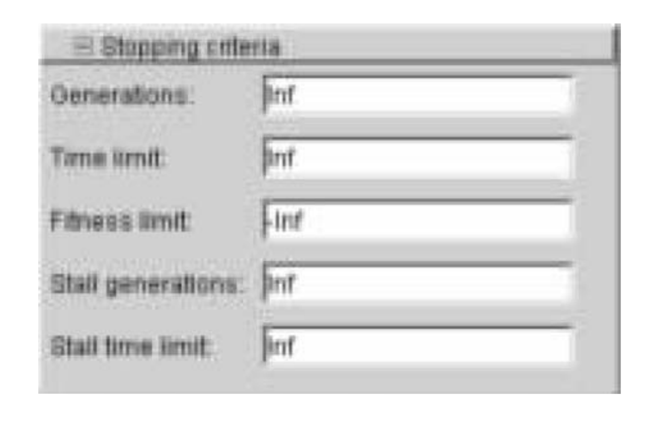



图 8.24 改变停止准则参数

**5. 图形显示**

图 8.25 所示为“Plots(绘图)”窗格，可以用来控制显示遗传算法运行结果变化的图形。

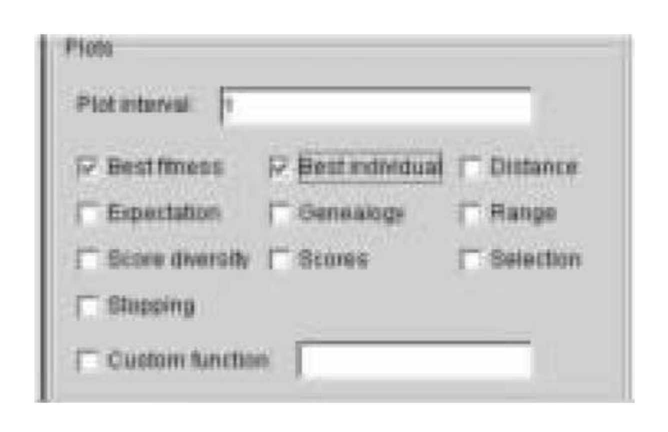



图 8.25 在绘图窗格中选择输出项

选择所要显示的图形参数的复选框。例如，如果选择“Best fitness(最佳适应度)”和“Best individual(最佳个体)”，运行例子“Rastrigin 函数”，其显示输出如图 8.26 所示。

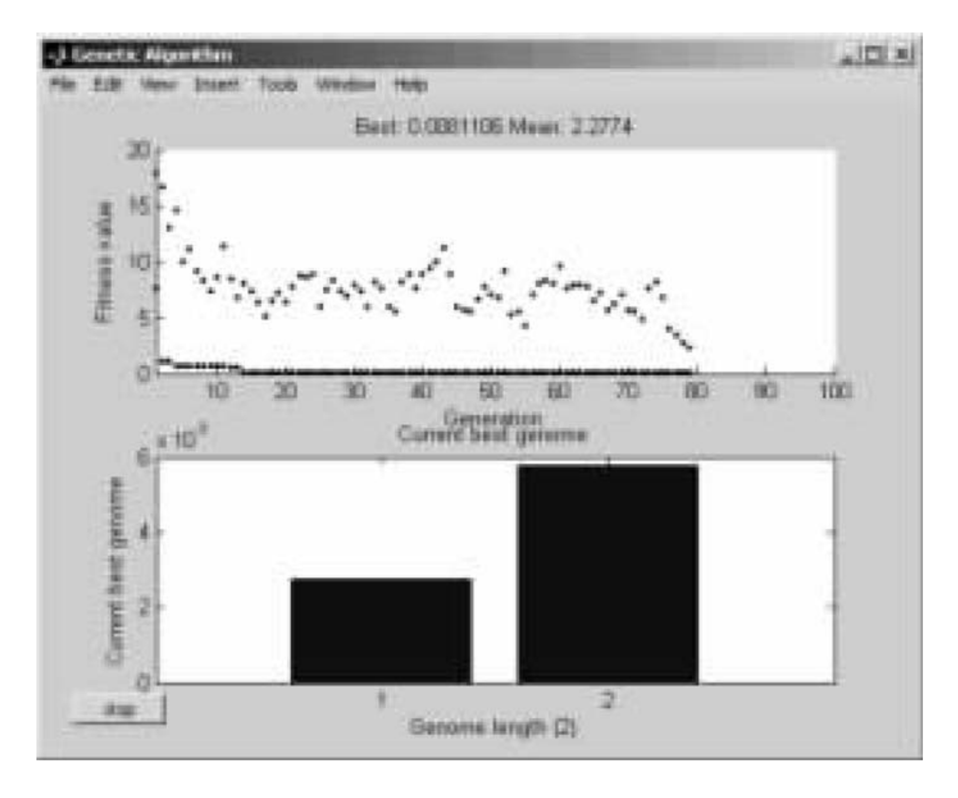



图 8.26 Rastrigin 函数最佳适应度与最佳个体

图 8.28 上部离散点为每一代的最佳适应度值和平均适应度值，下部柱型图表示当前代最佳适应度值对应的点的坐标。

注意：要想显示两个以上参数项的图形时，可选择相应参数项的复选框，单独打开一个较大的图形窗口即可。

**6．** 举例——创建用户绘图函数

如果工具箱中没有符合想要输出图形的绘图函数，用户可以编写自己的绘图函数。遗传算法在每次运行时调用这个函数，画出图形。这里举例说明怎样创建一个用户绘图函数来显示从前一代到当前代最佳适应度值的变化情形，内容包括：创建绘图函数，使用绘图函数，绘图函数如何作用。

1）创建绘图函数

为了创建绘图函数，在 MATLAB 编辑器中复制、粘贴下列代码到一个新的 M 文件。

```
 Function state=gaplotchange(options, state, flag)
% 绘图函数 gaplotchange 画出前一代个体最佳值变化的图形
persistent last_best                    % 前一代个体的最佳值
if (strcmp(flag,'init'))                % 绘图
     set(gca,'xlim',[1, options.Generations],'Yscale', 'log');
     hold on;
     xlabel Generation
     title('Change in Best Fitness Value')
end
best=min(state.Score);                  % 当前代个体的最佳值
if state.Generation == 0                % 设置 last_best 为最佳
     last_best=best;
else
     change=last_best - best;           % 改变个体的最佳值
     last_best=best;
     plot(state.Generation,change,'.r');
     title(['Change in Best Fitness Value'])
end
```

然后在 MATLAB 路径下将其存为 M 文件 gaplotchange.m。

2）使用绘图函数

为了使用用户绘图函数，在“Plots(绘图)”窗格中选择“Custom function(定制函数)”，并且在其右边的文本框中输入函数名“@gaplotchange”。为了对用户绘图函数输出的最佳适应度值图形进行比较，在这里也选择“Best fitness”。运行例子函数 Rastrigin，显示出来的图形如图 8.27 所示。

注意：因为图中下半部的 Y 轴为对数刻度，所以图形中的离散点仅仅显示大于零的点。对数刻度能显示适应度函数的微小变化，而图中上半部则不能显示出微小变化。

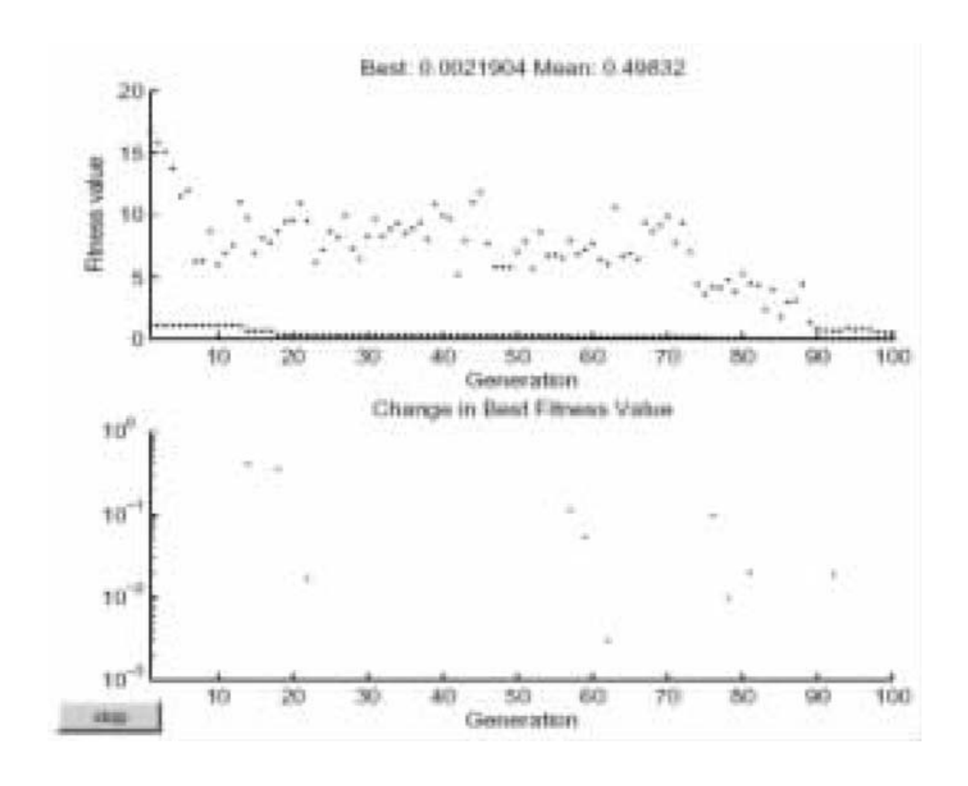



图 8.27　用户绘图函数输出的 Rastrigin 函数运行结果

3）绘图函数如何作用

绘图函数使用包含在下面结构体中的信息，它们由遗传算法传递给绘图函数作为输入参数：

（1）options（参数）：当前参数设置。

（2）state（状态）：关于当前代的信息。

（3）flag（曲线标志）：曲线表示为对数等的当前状态。

绘图函数的主要作用描述如下：

persistent last_best：生成永久变量 last_best ——即前一代的最佳值。永久变量保存着多种图形函数调用类型。

set(gca，'xlim'，[1，options.Generations]，'Yscale'，'log')：在遗传算法运行前建立图形。options.Generation 为代数的最大值。

best＝min(state.Score)：state.Score 包含当前代中所有个体的得分值，变量 best 是其中最小的得分值。结构体状态文本框的完整描述可参见 8.4.1 节“遗传算法参数”的“图形参数”部分。

change＝last_best － best：变量 change 是前一代的最佳值减去当前代的最佳值。

plot(state.Generation，change，'.r')：画出当前代的变化曲线，变量维数包含在 state.Generation 中。

函数 gaplotchange 的代码包含了函数 gaplotbestf 代码中许多相同成分，函数 gaplotbestf 生成最佳适应度图形。

**7. 复现运行结果**

为了复现遗传算法前一次的运行结果，选择“Use random states from previous run（使用前一次运行的随机状态）”复选框，这样就把遗传算法所用的随机数发生器的状态重新设置为前一次的值。如果没有改变遗传算法工具中的所有设置，那么遗传算法下一次运行时

返回的结果与前一次运行的结果一致。

正常情况下，不要选择此复选框，可以充分利用遗传算法随机搜索的优点。如果想要分析特定的运行结果或者显示相对个体的精确结果，可以选择此复选框。

**8．**设置选项参数

设置遗传算法工具使用时的选项参数有两种方法：一种是在遗传算法工具GUI的"Options"窗格中直接进行设置，另一种是在MATLAB工作窗口中通过命令行方式进行设置。

在"Options"窗格中设置遗传算法的各种运行参数，如图8.28所示。每一类参数对应有一个窗格，单击该类参数时，对应窗格展开。例如，单击"Population"参数选项，种群窗格展开来，可以逐一设置其中的参数项，如Population type(种群类型)、Population size(种群尺度)、Creation function(创建函数)、Initial population(初始种群)、Initial scores(初始得分)、Initial range(初始范围)等。

图8.28　选项参数窗口

此外，其他选项参数类还有：Fitness scaling(适应度测量)、Selection(选择)、Reproduction(复制)、Mutation(变异)、Crossover(交叉)、Migration(迁移)、Hybrid function(混合函数)、Stopping criteria(停止准则)、Output function(输出函数)、Display to

command window(显示到命令窗口)、Vectorize(向量化)等。这些参数类各自对应一个参数窗格，单击后相应窗格随即展开，在其中可以进行参数项的设置。

所有变量参数的含义及详细描述可参见8.4.1节“遗传算法参数”。

在MATLAB工作窗口中，可以将遗传算法的运行参数设置为变量。

对于数值参数的设置，可以直接在相应编辑框中输入该参数的值，或者在包含该参数值的MATLAB工作窗口中输入相应变量的名字，就可以完成设置。例如，可以利用下面两种方法之一设置“Initial point(初始点)”为[2.1  1.7]：

(1) 在“Initial point”文本框中输入[2.1  1.7]。

(2) 在MATLAB工作区中输入变量x0=[2.1  1.7]，然后在“Initial point”文本框中输入变量的名字x0。

因为选项参数是比较大的矩阵或向量，所以在MATLAB工作窗口中把参数的值定义为变量一般是比较方便的，也就是说，如果需要，很容易改变矩阵或向量的项。

**9. 输入输出参数及问题**

参数和问题结构可以从遗传算法工具被输出到MATLAB的工作窗口，也可以在以后的某个时间再反过来把它们从MATLAB的工作窗口输入给遗传算法工具。这样就可以保存对问题的当前设置，并可以在以后恢复这些设置。参数结构也可以被输出到MATLAB工作窗口，并且可以再把它们用于命令行方式的遗传算法函数ga。

输入和输出信息通常包含下列各项：

(1) 问题定义，包括“Fitness function(适应度函数)”和“Number of variables(变量个数)”。

(2) 当前指定的选项参数。

(3) 算法运行的结果。

下面解释如何输入和输出这些信息。

1) 输出参数和问题

参数和问题可以被输出到MATLAB工作空间，以便以后在遗传算法工具中应用。也可以以命令行方式，在函数ga中应用这些参数和问题。

为了输出参数和问题，单击“Export to Workspace(输出到工作空间)”按钮或从“File”菜单中选择“Export to Workspace”菜单项，打开如图8.29所示的对话框。

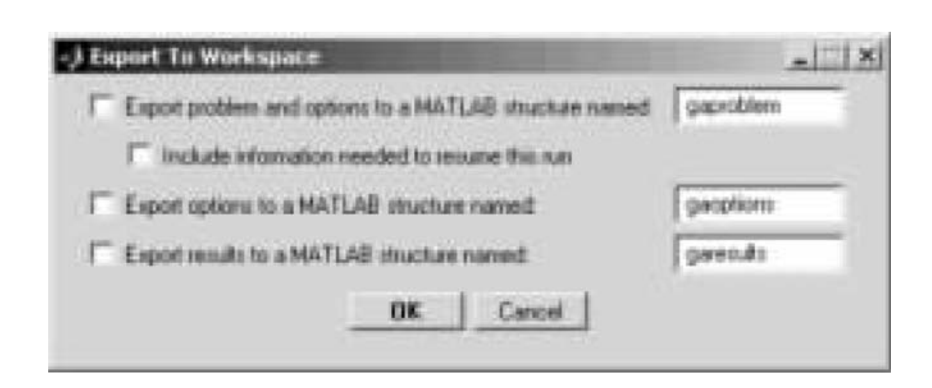



图8.29  输出对话框

对话框提供下列参数：

(1) 为了保存问题的定义和当前参数的设置，选择“Export problem and options to a

MATLAB structure named(输出问题与参数到已命名的 MATLAB 结构)”，并为这个结构体命名。单击“OK”按钮，即把这个信息保存到 MATLAB 工作空间的一个结构体。如果以后要把这个结构体输入到遗传算法工具，那么当输出这个结构时，所设置的“Fitness function”和“Number of variables”以及所有的参数设置都被恢复到原来值。

注意：输出问题之前，如果在“Run solver(运行求解器)”窗格中选中“Use random states from previous run(使用前一次运行的随机状态)”选项，则遗传算法工具将保存上一次运行开始时随机数产生函数 rand 和 randn 的状态。然后，在选择了此选项的情况下，输入问题且运行遗传算法，那么输出问题之前的运行结果就被准确地复现了。

(2) 如果想要遗传算法在输出问题之前从上一次运行的最后种群恢复运行，可选择“Include information needed to resume this run(包括所需信息以恢复本次运行)”。然后，当输入问题结构体并单击“Start”按钮，算法就从前次运行的最后种群继续运行。为了恢复遗传算法产生随机初始种群的缺省行为，可删除在“Initial population”字段所设置的种群，并用代之以空的中括号“[ ]”。

注意：当选择了“Include information needed to resume this run”选项，则选择“Use random states from previous run”选项对于输入问题和运行遗传算法时创建的初始种群将不再有任何作用。后一个选项只是指定从新的一次运行开始时再一次复现结果，而不是为了恢复算法的继续运行。

(3) 如果只是为了保存参数设置，可选择“Export options to a MATLAB structure named(输出参数到已命名的 MATLAB 结构)”，并为这个参数结构体输入一个名字。

(4) 为了保存遗传算法最近一次运行的结果，可选择“Export results to a MATLAB structure named”，并为这个结果结构体命名。

2) 举例——用输出问题运行函数 ga

输出一个问题可参见 8.2.3 节“举例：Rastrigin 函数”，在命令行运行遗传算法函数 ga，其步骤如下：

(1) 单击“Export to Workspace”按钮，打开相应对话框。

(2) 在“Export To Workspace”对话框中的“Export problem and options to a MATLAB structure named”右边的文本框中，输入问题结构体的名称，假设为 my_gaproblem。

(3) 在 MATLAB 窗口中以 my_gaproblem 为参数调用函数 ga：

```
[x fval]=ga(my_gaproblem)
```

则返回结果：

```
x=
    0.0027 -0.0052
fval=
    0.0068
```

调用函数 ga 可参见 8.3.2 节“从命令行使用遗传算法”。

3) 输入参数

为了从 MATLAB 工作窗中输入一个参数结构体，可从“File”菜单选择“Import Options(输入参数)”菜单项。在 MATLAB 工作窗口中打开一个对话框，列出遗传算法参数结构体的一系列选项。当选择参数结构体并单击“Import(输入)”按钮时，在遗传算法工

具中的参数域就被更新，且显示所输入参数的值。

创建参数结构体有两种方法：

（1）调用函数 gaoptimset，以参数结构 options 作为输出。

（2）在遗传算法工具中的“Export to Workspace(输出到工作空间)”对话框中保存当前参数。

4）输入问题

为了从遗传算法工具输入一个以前输出的问题，可从“File”菜单选择“Import Problem(输入问题)”菜单项。在 MATLAB 工作窗口中，打开一个对话框，显示遗传算法问题结构体的一个列表。当选择了问题结构体并单击“OK”按钮时，遗传算法工具中的下列文本框被更新：

（1）适应度函数。

（2）变量个数。

（3）参数域。

**10. 举例——从最后种群中继续遗传算法**

下面的例子显示如何输出一个问题，以便当输入问题并单击“Start”按钮时，遗传算法能从该输出问题所保存的最后种群继续运行。现在运行一个例子，在遗传算法工具中输入下面的信息：

（1）设置适应度函数为@ackleyfcn，它是计算函数 Ackley，是工具箱提供的一个测试函数。

（2）设置“Number of variables”为 10。

（3）在“Plots”窗格中选择“Best fitness”。

（4）单击“Start”按钮。

显示的结果如图 8.30 所示。

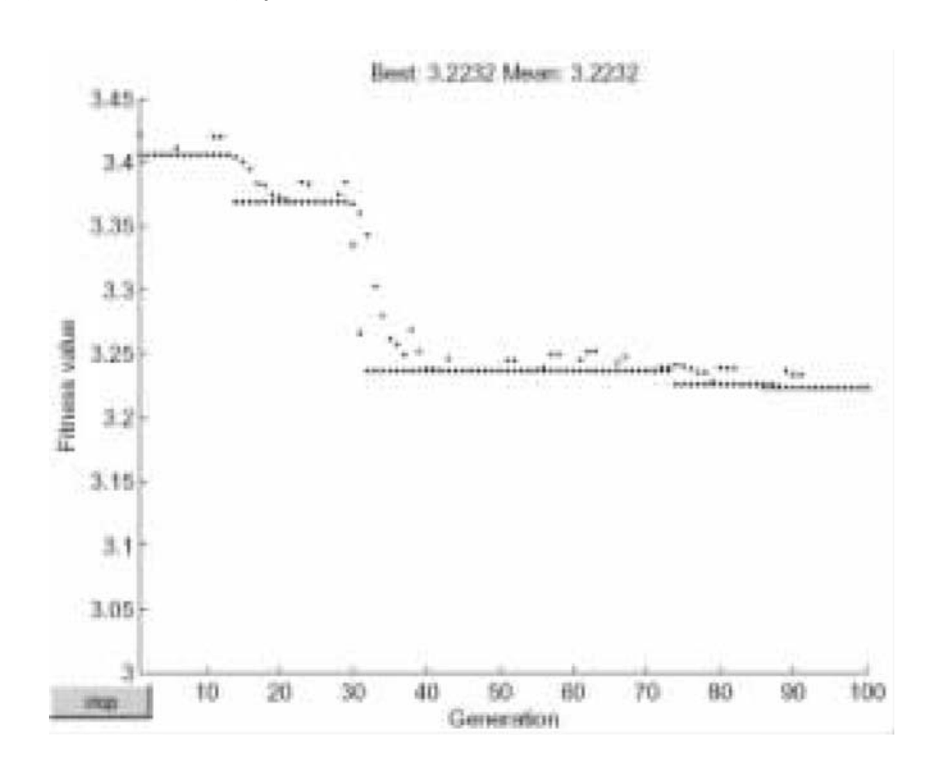



图 8.30　函数 ackleyfcn 的最佳适应度

假定想要实验利用其他的参数运行遗传算法，接着利用当前参数设置，此后再从最后种群重新运行算法。为此，进行以下步骤：

（1）单击“Export to Workspace”按钮。

（2）在出现的对话框中：

① 选择“Export problem and options to a MATLAB structure named”。

② 在文本框中输入问题和参数的名称，如 ackley_uniform。

③ 选择“Include information needed to resume this run（包括所需信息以恢复本次运行）”。

做了这些选择后，打开如图 8.31 所示的对话框。

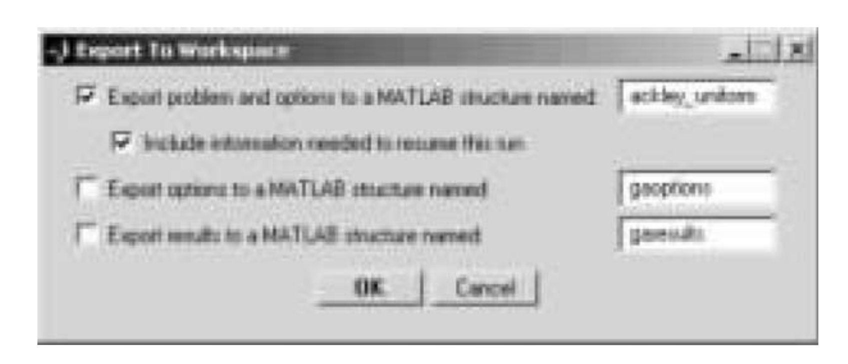



图 8.31 在输出窗口对话框中做适当选择

（3）单击“OK”按钮。

问题和参数被输出到 MATLAB 工作空间的一个结构体中。在 MATLAB 命令窗口中输入下面的信息就可以观察这个结构体：

```
ackley_uniform
ackley_uniform =
    fitnessfcn: @ackleyfcn
    genomelength: 10
    options: [1x1 struct]
```

利用不同的参数设置，甚至是不同的适应度函数，在运行遗传算法之后，都能够按照如下步骤来恢复问题：

（1）从“File”菜单中选择“Import Problem ”菜单项，打开如图 8.32 所示的对话框。




图 8.32 GA 问题输入窗口

（2）选择 ackley_uniform。

（3）单击“Import”按钮。

这样就把“Population”选项中的“Initial population”字段设置成输出问题之前运行的最后种群。在运行期间，所有其他参数恢复它们的设置。当单击“Start”按钮时，遗传算法从被保存的最后种群重新运行。图 8.33 所示为初始运行和重新运行的最佳适应度图形。

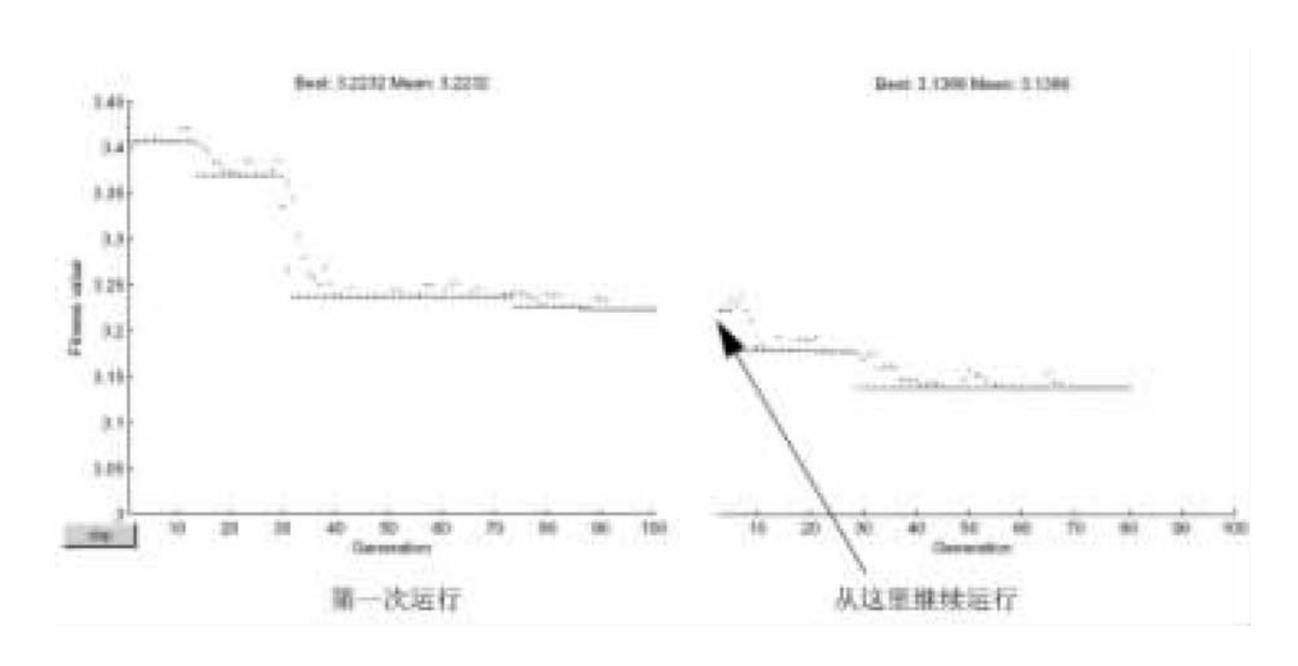



图 8.33　初始运行和重新运行的最佳适应度

注意：如果在利用所导入问题运行遗传算法之后，想要恢复遗传算法生成一个随机初始种群的缺省行为，可删除“Initial population”字段中设置的种群，而代之以空的中括号“[ ]”。

**11.** 生成 **M** 文件

在遗传算法工具中，要利用特定的适应度函数和参数生成运行遗传算法的 M 文件，可从“File”菜单中选择“Generate M-File(生成 M 文件)”菜单项，并把生成的 M 文件保存到 MATLAB 路径的一个目录。

在命令行调用这个 M 文件时，返回的结果与利用在遗传算法工具中生成 M 文件时的适应度函数和参数所得到的结果一致。

## 8.3.2　从命令行使用遗传算法

使用遗传算法，也可以从命令行运行遗传算法函数 ga。这方面的内容主要包括：利用缺省参数运行 ga；在命令行设置 ga 的参数；使用遗传算法工具的参数和问题结构；复现运行结果；以前一次运行的最后种群重新调用函数 ga；从 M 文件运行 ga。

**1.** 利用缺省参数运行 **ga**

利用缺省参数运行遗传算法，以下面的语句调用 ga：

```
[x fval]=ga(@fitnessfun,nvars)
```

其中，@fitnessfun 是计算适应度函数值的 M 文件的函数句柄；nvars 是适应度函数中独立变量的个数；x 是返回的最终点；fval 是返回的适应度函数在 x 点的值。

例如，运行 Rastrigin 函数，从命令行输入

```
[x fval]=ga(@rastriginsfcn,2)
```

将返回

```
x=
    0.0027    -0.0052
fval=
    0.0068
```

为了得到遗传算法更多的输出结果，可以用下面的语句调用 ga：

```
[x fval reason output population scores]=ga(@fitnessfcn, nvars)
```

除了输出变量 x 和 fval 之外，增加了以下输出变量：

(1)“reason(原因)”：算法停止的原因。

(2)“output(输出)”：包含关于算法在每一代性能的结构体。

(3)“population(种群)”：最后种群。

(4)“scores(得分)”：最终得分值。

**2．在命令行设置 ga 的参数**

遗传算法工具中的参数可以指定为任何有效的参数值，设置参数使用下面的语句：

```
[x fval]=ga(@fitnessfun,nvars,options)
```

使用函数 gaoptimset 生成一个参数结构体：

```
options=gaoptimset
```

返回带有缺省值的参数结构体：

```
options=
PopulationType:          'doubleVector'
PopInitRange:            [2x1 double]
PopulationSize:          20
EliteCount:              2
CrossoverFraction:       0.8000
MigrationDirection:      'forward'
MigrationInterval:       20
MigrationFraction:       0.2000
Generations:             100
TimeLimit:               Inf
fitnessLimit:            -Inf
StallLimitG:             50
StallLimitS:             20
InitialPopulation:       [ ]
InitialScores:           [ ]
PlotInterval:            1
CreationFcn:             @gacreationuniform
fitnessScalingFcn:       @fitscalingrank
SelectionFcn:            @selectionstochunif
```

```
CrossoverFcn:        @crossoverscattered
MutationFcn:         @mutationgaussian
HybridFcn:           [ ]
Display:             'final'
PlotFcns:            [ ]
OutputFcns:          [ ]
Vectorized:          'off'
```

如果没有给某一参数项输入新的值，则函数 ga 使用其缺省值。

每一个参数的值都存放在参数结构体中，例如 options.PopulationSize，可以通过输入参数的名称显示参数的值。例如，显示遗传算法种群的大小，可输入

```
options.PopulationSize
ans=
    20
```

改变参数结构体中成员值，例如，设置 PopulationSize 值等于 100，代替它的缺省值 20，可输入

```
options=gaoptimset('PopulationSize',100)
```

参数结构体中，PopulationSize 为 100，其他值都为缺省值或当前值。

这时，再输入

```
ga(@fitnessfun,nvars,options)
```

函数 ga 将以种群中个体为 100 运行遗传算法。

如果想接着改变参数结构体其他成员的值，例如设置 PlotFcns 为@gaplotbestf，目的是画出每一代最佳适应度函数值图形，则可用下面的语句调用函数 gaoptimset：

```
options=gaoptimset(options,'PlotFcns',@plotbestf)
```

这里保持了参数的所有当前值，除 PlotFcns 之外，它改变为@plotbestf。注意，如果省略输入自变量参数 options，那么函数 gaoptimset 重新置 PopulationSize 为它的缺省值 20。

也可以利用一个语句来同时设置两个参数 PopulationSize 和 PlotFcns：

```
options=gaoptimset('PopulationSize',100,'PlotFcns',@plotbestf)
```

**3.** 使用遗传算法工具的参数和问题结构

利用函数 gaoptimset 创建一个参数结构体，在遗传算法工具中设置参数的值，然后在 MATLAB 工作窗口中输出参数给结构体。如果想在遗传算法工具中输出缺省值，则导出的结构体的参数与由命令行得到的缺省结构体的参数一致。

```
options=gaoptimset
```

如果想从遗传算法工具输出一个问题结构体 ga_problem，可用下面的语句调用函数 ga：

```
[x fval]=ga(ga_problem)
```

问题结构体包含以下参数：

(1) fitnessfcn：适应度函数。

(2) nvars：问题的变量数。

(3) options：参数结构体。

**4．** 复现运行结果

因为遗传算法是随机性方法，也就是说，产生随机机率，即每次运行遗传算法得到的结果都会略有不同。遗传算法利用 MATLAB 随机数产生器函数 rand 和 randn，在每一次迭代中产生随机机率。每一次函数 ga 调用 rand 和 randn，它们的状态都可能发生改变，以便下一次再被调用时返回不同的随机数。这就是每次运行后 ga 输出的结果会略有不同的原因。

如果需要准确复现运行结果，可以在调用函数 ga 时包含 rand 和 randn 的当前状态。在又一次运行 ga 之前，重新设置这些值的状态。例如，要复现 Rastrigin 函数的 ga 的输出，可以利用下面的语句调用 ga：

```
[x fval reason output]=ga(@rastriginsfcn,2);
```

假设某次运行的返回结果为

```
x=
    0.0027    -0.0052
fval=
    0.0068
```

则随机函数 rand 和 randn 两者的状态被保存在 output 结构中：

```
output=
    randstate:[35x1 double]
    randnstate:[2x1 double]
    generations:100
    funccount:2000
    message:[1x64 char]
```

然后，重新设置状态，输入

```
rand('state',output.randstate);
randn('state',output.randnstate);
```

如果现在再次运行 ga，就会得到相同的结果。

注意：如果没有必要复现运行结果，最好不要设置 rand 和 randn 的状态，以便能够得到遗传算法随机搜索的益处。

**5．以前一次运行的最后种群重新调用函数 ga**

缺省情况下，每次运行 ga 时都生成一个初始种群。然而，可以将前一次运行得到的最后种群作为下一次运行的初始种群，这样做能够得到更好的结果。这可以利用下面的语句实现：

```
[x,fval,reason,output,final_pop]=ga(@fitnessfcn,nvars);
```

最后一个输出变量 final_pop 返回的就是本次运行得到的最后种群。将 final_pop 再作为初始种群运行 ga，实现语句为：

```
options=gaoptimset('InitialPop',final_pop);
[x,fval,reason,output,final_pop2]=ga(@fitnessfcn,nvars);
```

还可以将第二次运行 ga 得到的最后种群 final_pop2 作为第三次运行 ga 的初始种群。

**6. 从 M 文件运行 ga**

利用命令行可以运行遗传算法。使用 M 文件可以有不同的参数设置。例如，可以设置不同的交叉概率来运行遗传算法，观察、比较每次运行的结果。下面的代码是运行 ga 函数 21 次，变量 options.CrossoverFraction 从 0 到 1，间隔为 0.05，所记录的运行结果。

```
options=gaoptimset('Generations',300);
rand('state',71);        % 这两个命令仅仅使结构成为可再现的
randn('state',59);
record=[ ];
for n=0:.05 :1
    options=gaoptimset(options,'CrossoverFraction',n);
    [x fval]=ga(@rastriginsfcn,10,options);
    record=[record; fval];
end
```

可以利用下列语句，以不同概率画出 fval 值的曲线图形：

```
plot(0 :.05 :1,record);
xlabel('Crossover Fraction');
ylabel('fval')
```

显示结果如图 8.34 所示。

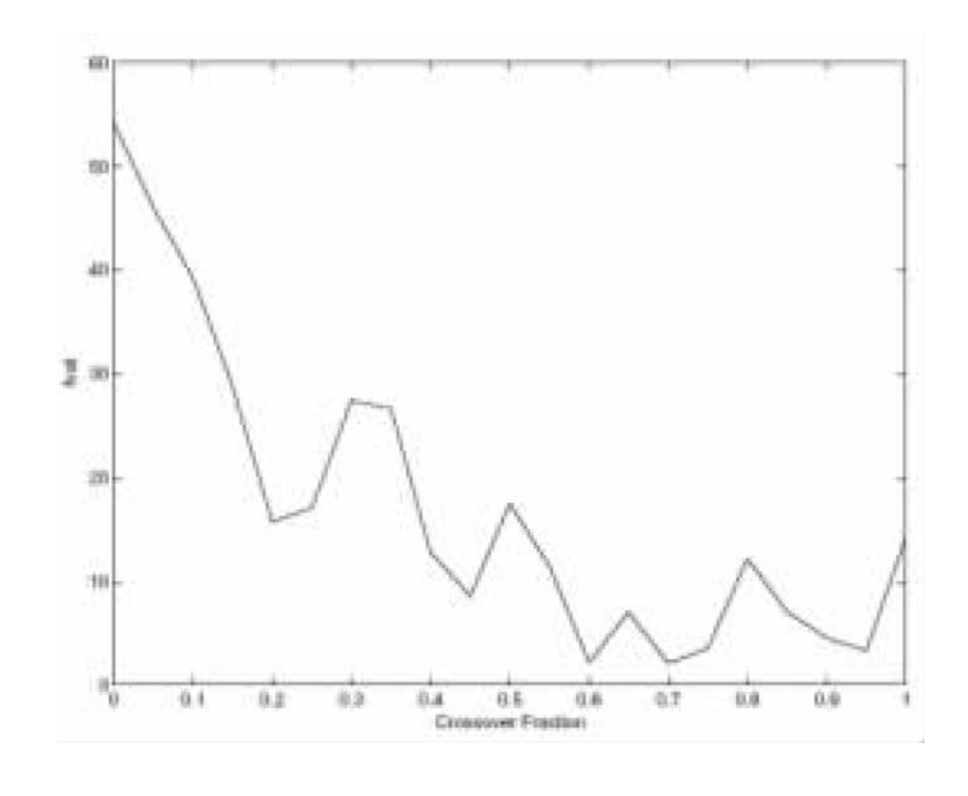



图 8.34　从 M 文件运行遗传算法时 fval 值的曲线图形

从图形显示可以看出，options.CrossoverFraction 的值为 0.6～0.95 时，可得到最好结果。

取每次运行得到的 fval 的平均值，就可以画出 fval 的光滑曲线，如图 8.35 所示。曲线最凹的部分对应 options.CrossoverFraction 的值为 0.7～0.9。

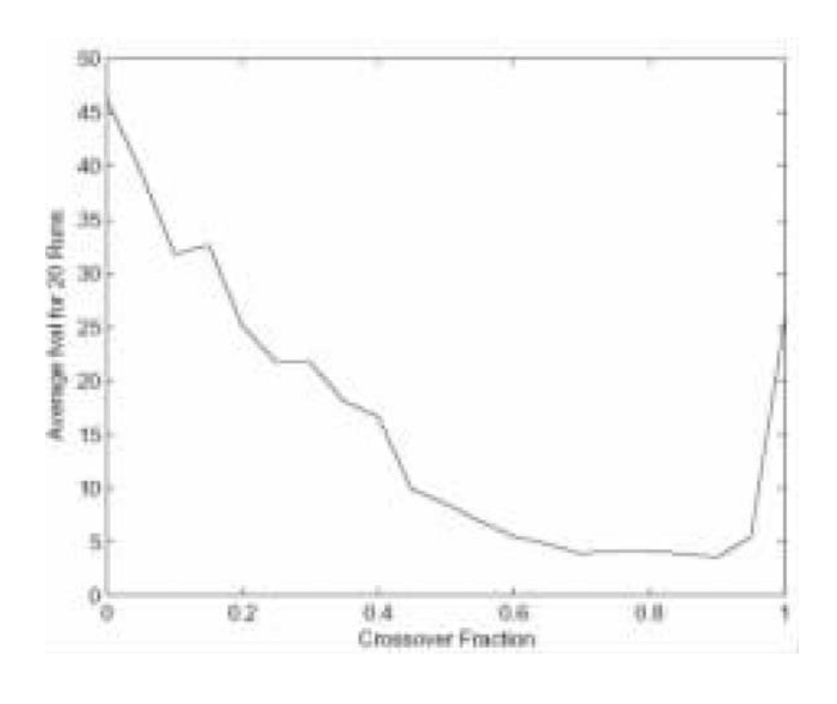



图 8.35　从 M 文件运行遗传算法时 fval 平均值的曲线图形

### 8.3.3　遗传算法举例

为了得到遗传算法的最好结果，一般需要以不同的参数试验。通过不断试验，选择针对问题的最佳参数。有效参数的完整描述可参见 8.4.1 节“遗传算法参数”。

本节介绍几种能够提高运算效果的参数改变方法，内容包括：种群多样性、适应度测量、选择、复制参数、变异与交叉、设置变异大小、设置交叉概率、相对于全局的局部最小值、使用混合函数、设置最大代数和向量化适应度函数。

**1．**种群的多样性

决定遗传算法的一个重要性能是种群的多样性。个体之间的距离越大，则多样性越高；反之，个体之间的距离越小，则多样性越低。由试验可得到种群的适当多样性。如果多样性过高或者过低，遗传算法都可能运行不好。这里介绍如何设置种群的初始范围来控制种群的多样性，并介绍如何设置种群尺度。

1）举例——设置初始范围

遗传算法工具在默认情况下利用生成函数随机生成一个初始种群。使用者可以在“Population”的“Initial range”文本框中指定初始种群的向量范围。

注意：初始范围仅仅限制在初始种群中的点的范围。后续各代包含的点可以不在初始种群的范围之内。

如果知道问题解的大概范围，计算时就可以指定包含问题解的初始范围。但是，假设种群具有足够的多样性，遗传算法就可以找到不在初始范围的解。下面的例子显示初始范围对遗传算法性能的影响。这个例子利用 Rastrigin 函数，函数在原点处取得最小值为 0。运行之前在遗传算法工具中设置下列参数：

（1）设置适应度函数为 @Rastriginsfcn。

（2）设置“Number of variables”为 2。

（3）在“Plots”窗格中选择“Best fitness”。

（4）在“Plots”窗格中选择“Range”。

（5）设置“Initial range”为［1；1.1］。

然后，单击“Start”按钮，遗传算法返回最佳适应度值为 2，其显示图形如图 8.36 所示。

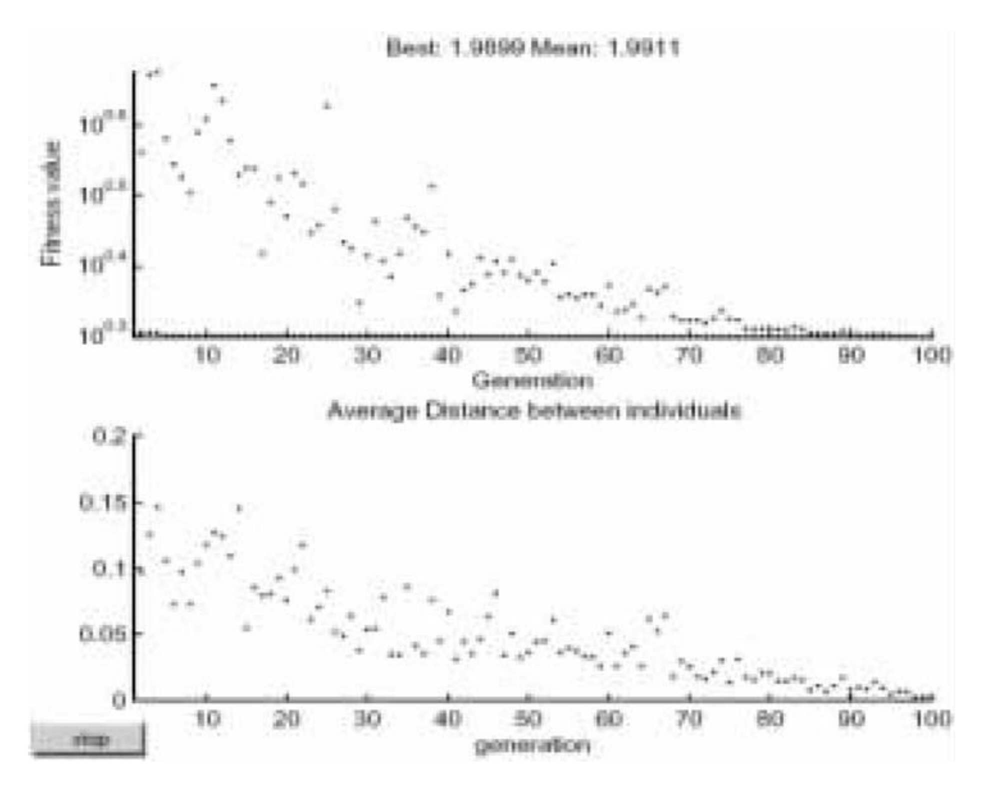



图 8.36　初始范围为[1；1.1]时的最佳适应度值和平均距离

图 8.38 上部为每代最佳适应度值变化图，下部为每代个体之间平均距离图，它可以很好地用来衡量种群的多样性。对于初始范围的设置，由于多样性太小，算法进展很小。

第二次，尝试设置“Initial range”为［1；100］，运行算法，得到最佳适应度值大约为 3.9，如图 8.37 所示。

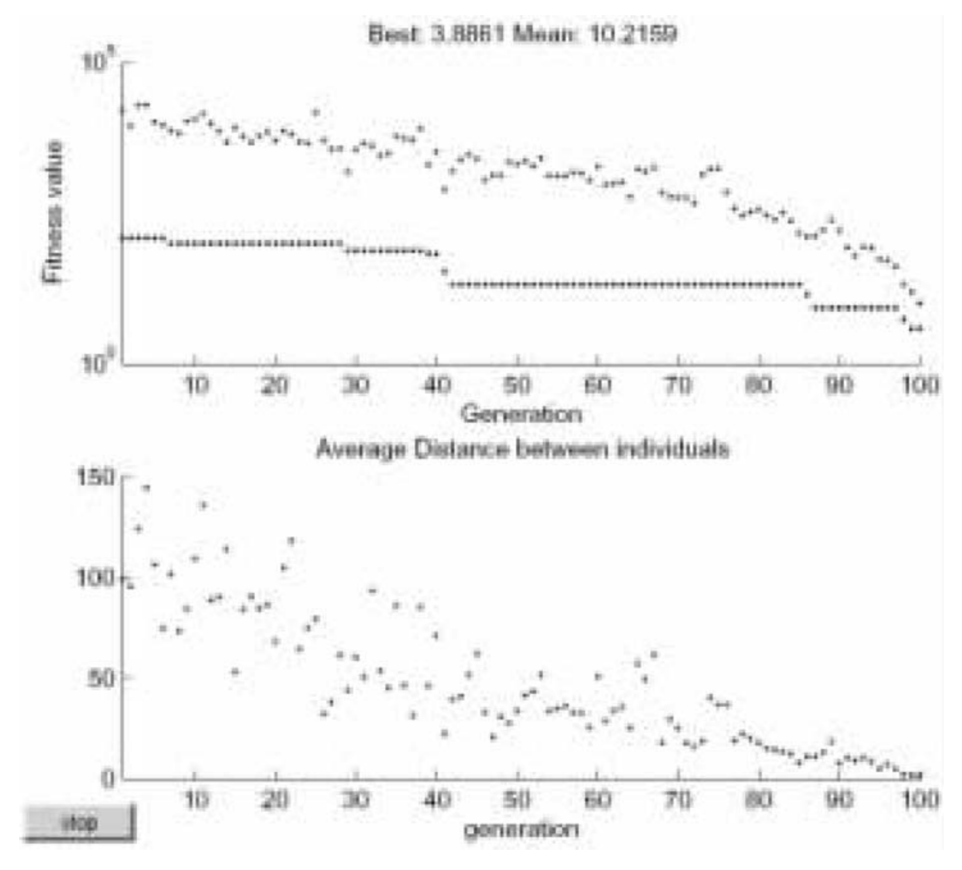



图 8.37　初始范围为[1；100]时的最佳适应度值和平均距离

这次，算法进展较快。但是，由于个体之间的平均距离太大，最佳个体远离最优解。

第三次，设置“Initial range”为［1；2］，运行算法，得到最佳适应度值大约为 0.012，如图 8.38 所示。

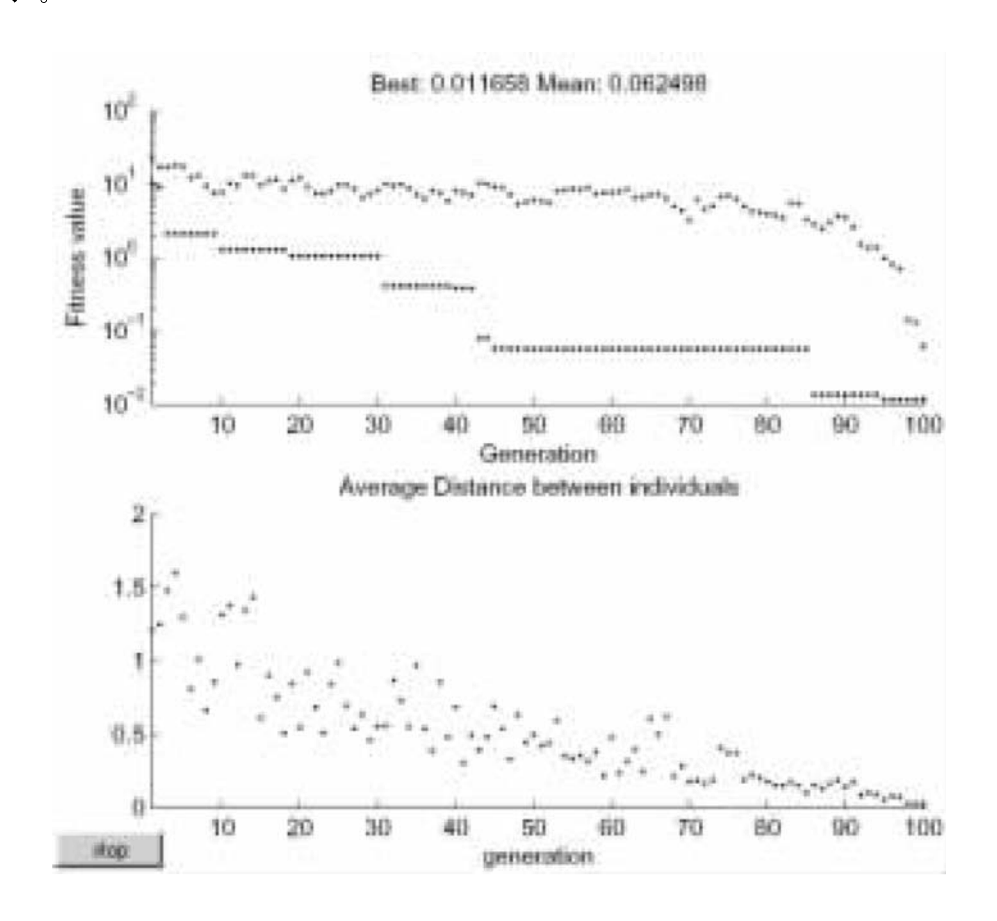



图 8.38　初始范围为[1；2]时的最佳适应度值和平均距离

这次由于多样性比较适合这个问题，所以算法得到的结果比前两次都好。

2）设置种群尺度

在“种群(Population)”参数域中“Size”决定每代种群的大小。增加种群的大小，遗传算法能够搜索更多的点，因此，能够得到较好结果。但是，种群越大，遗传算法每代运行时间越长。

注意：至少应该设置“尺度(Size)”的值为“Number of variables”，以便在每一种群中使个体超出搜索范围。

进行实验时，可以设置不同的种群尺度，不限制运行时间，以期得到最好结果。

**2．** 适应度测量

适应度测量把适应度函数返回的原始适应度得分值转换为适合选择函数范围内的值。选择函数根据适应度值的大小，选择下一代的父体。选择函数分配大选择概率给适应度值大的个体。适应度测量值的范围影响遗传算法的性能。如果测量值变化范围太大，则具有高测量值的个体复制的速度很快，取代种群基因池的速度很快，妨碍了遗传算法搜索解空间的其他区域。相反，如果测量值变化太小，所有个体复制机会基本相同，则搜索过程进展缓慢。

缺省的适应度尺度变换函数为 Rank(排序)，根据每个个体的顺序而不是它的得分值来变换原始得分值。个体的顺序是它在排序后的位置。最适应的个体的序号为 1，次之为 2，依次类推。排序尺度变换函数分配尺度值有下列目的：

(1) 个体的尺度值与 n 成正比。

(2) 整个种群的尺度值的和等于要求生成下一代父体的数目。

(3) 排序适应度尺度变换函数避免了初始值的界限的影响。

图 8.39 所示为具有 20 个个体的一个典型种群的初始得分值(按升序排序)。

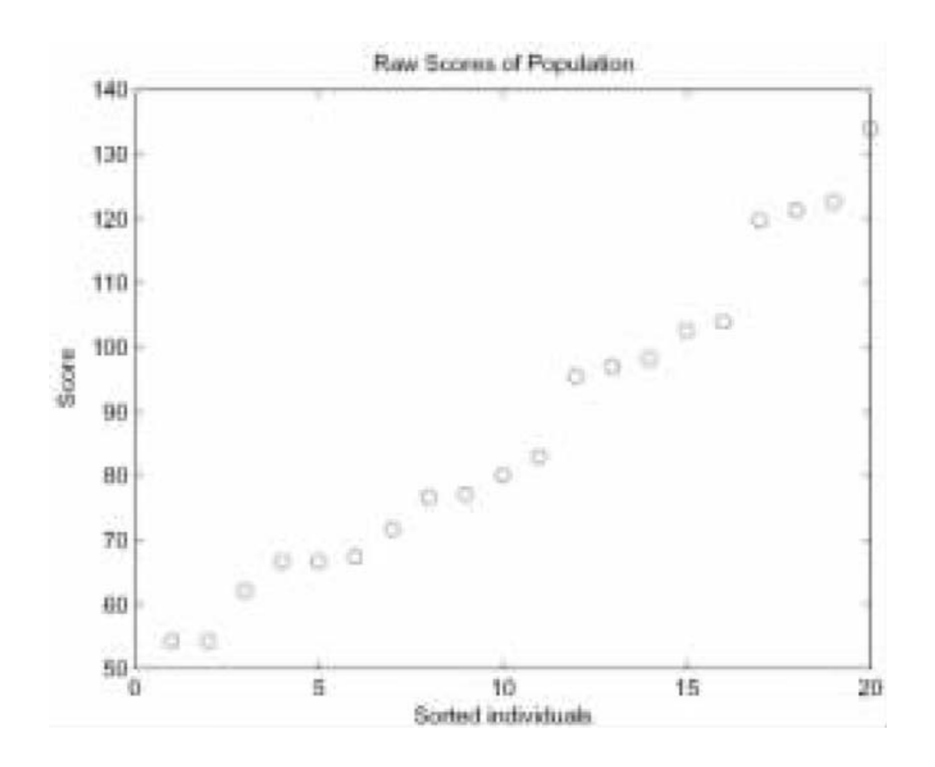



图 8.39 具有 20 个个体的一个典型种群的初始得分值

图 8.40 所示为使用尺度变换函数的初始尺度值。

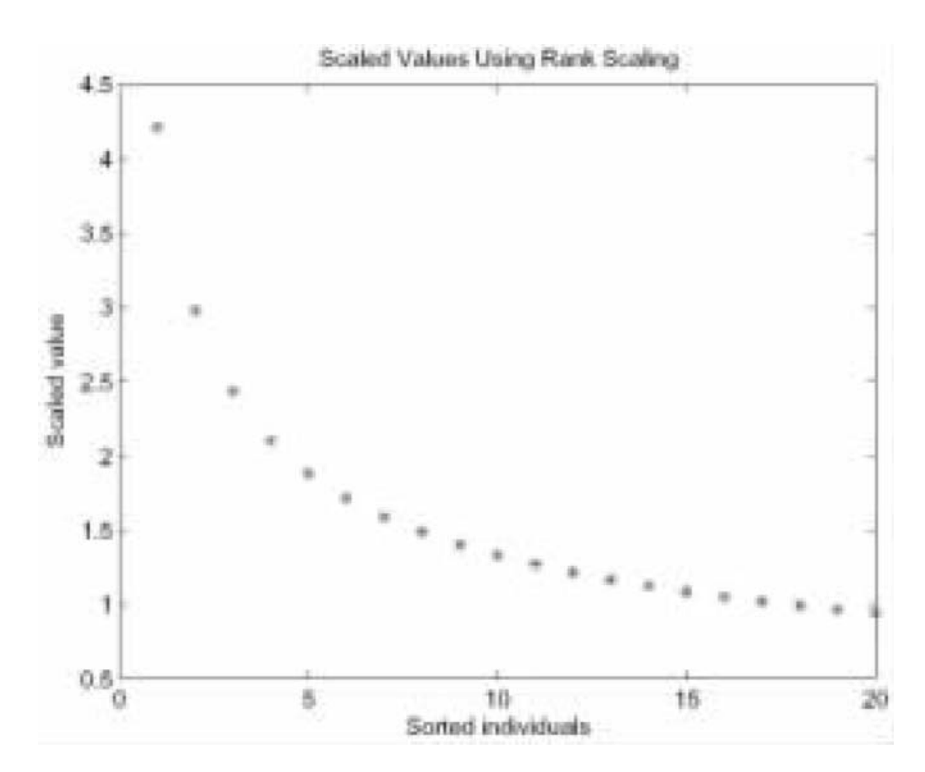



图 8.40 使用尺度变换函数的初始尺度值

因为算法按适应度函数的最小化处理,所以低的初始值具有高的尺度值。又因为排序尺度变换只根据个体的顺序分配值的大小,对于一个大小为 20、父辈数等于 32 的群体,显示的尺度值可以是相同的。

可以把排序尺度变换(Rank scaling)与顶级尺度变换(Top scaling)进行比较。为了观察尺度变换的效果，可以把遗传算法利用排序尺度变换得到的结果与利用其他函数(如顶级变换)得到的结果相比较。默认情况下，顶级尺度变换分配4个最佳适应度个体相同的尺度值，等于父辈数除以4，而分配其他个体的尺度值为0。利用默认的选择函数，只有4个最佳适应度个体能被选为父辈。图8.41所示为比较排序尺度变换与顶级尺度变换得到的尺度值，种群尺度为20，父辈数为32。

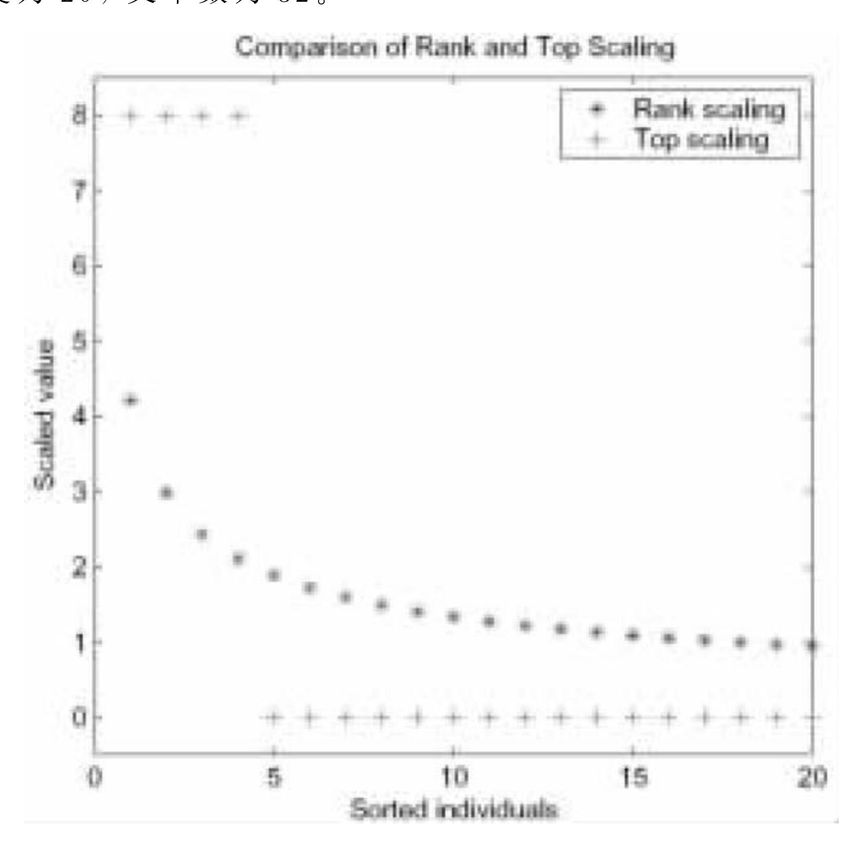



图8.41 比较排序尺度变换与顶级尺度变换得到的尺度值

因为Top scaling限制父辈为最佳适应度个体，它产生的种群类型比Rank scaling产生的种群类型少。图8.42所示为每一代Rank scaling与Top scaling得到的个体之间的距离变化的比较。

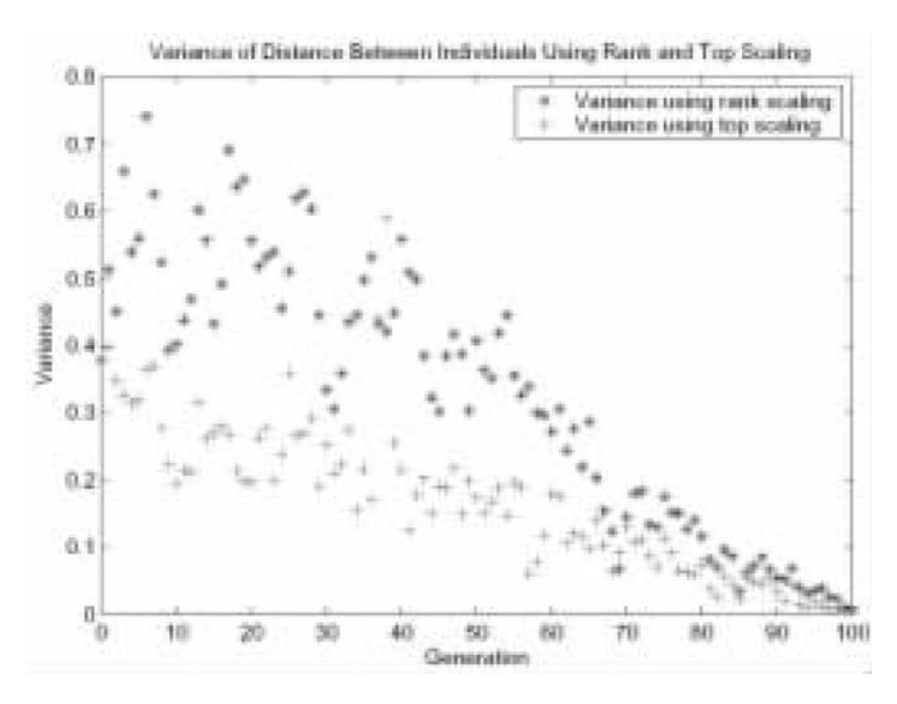



图8.42 排序尺度变换与顶级尺度变换各代个体之间距离变化之比较

**3. 选择**

选择函数根据个体由适应度尺度变换函数得到的尺度值，为下一代选择父辈。当一个个体为多个子辈贡献它的基因时，它就可能多次被选做父辈。默认的选择函数为Stochastic uniform(随机均匀函数)——在每一父辈画出一条与选择线对应的直线，长度与它的尺度值成比例。算法以等步长在线上移动。在每一步中，算法从落入线上的部分分配父辈。

一个比较确定的选择函数是 Remainder，由下列两步运行：

(1) 函数按照尺度值的整数部分为每个个体选择父辈。例如，假设一个个体的尺度值是 2.3，函数选择这个个体两次作为父辈。

(2) 在随机均匀选择时，选择函数利用尺度值的小数部分选择剩余的父辈。函数落入选择线内，即长度与个体尺度值的小数部分成比例，在线上按等步长移动来选择父辈。注意，如果尺度值的小数部分都等于 0，则就像顶级尺度变换一样，选择是完全确定的。

**4. 复制参数**

复制参数控制遗传算法怎样生成下一代。这些参数有：

(1) Elite count(优良计数)：在当前种群中，具有最佳适应度值的个体遗传到下一代的个体数。这些个体称为优良子辈(Elite children)。Elite count 的默认值为 2。当优良计数至少为 1 时，最佳适应度值可能从一代到下一代减少。这是人们希望的，因为遗传算法使适应度函数最小化。设置 Elite count 为一个大数，可以使得最适应个体控制种群，但可能减小搜索的有效性。

(2) Crossover fraction(交叉概率)：下一代个体的一小部分，它不是 Elite children(优良子辈)，而是由交叉产生的部分。参见本节"设置交叉概率"部分的内容。

**5. 变异与交叉**

遗传算法运用当前代的个体生成子代个体，构成下一代。除了 Elite children 外，算法还生成下列子代个体：

(1) 从当前代中选择两个个体，交换两个个体的某个或某些位(基因)，结合后形成交叉子个体。

(2) 随机改变当前代的单个个体，形成变异子个体。

这两个过程是遗传算法的主要过程。交叉能够使遗传算法从不同的个体中提取更好的基因，结合到具有优势的子个体中。变异增加了种群的多样性，因而增大了算法生成更高适应度值的个体的可能性。没有变异，算法只能产生由初始种群结合基因的子集构成的个体。

算法生成的子个体类型如下：

(1) Elite count：在"Reproduction"文本框中指定 Elite children 的数目。

(2) Crossover fraction：在"Reproduction"文本框中指定种群中交叉子个体的概率，它不同于 Elite children。例如，假设 Population size(种群尺度)为 20，Elite count 为 2，Crossover fraction 为 0.8，则下一代子个体类型如下：

① 有 2 个优良子辈。

② 除优良子辈以外，还有 18 个个体。所以计算 $0.8\times18=14.4$ 取整得 14，得到 14 个交叉子个体。

③ 还有 4 个个体，它们不是优良子辈，而是变异子个体。

**6．** 设置变异大小

遗传算法应用变异函数(Mutation function)字段中指定的函数进行变异操作。默认的变异函数为高斯函数 Gaussian，它把一个从高斯分布选择的随机数(即 mutation)，加到父辈向量的每一个项上。典型情况下，与分布的标准差成比例的变异大小，在每一后代中都是减小的。通过参数尺度(Scale)和压缩(Shrink)，可以控制每一代变异的平均数量。

尺度控制第一代变异的标准差。标准差是 Scale 乘以初始种群的范围，该范围是使用者由 Initial range 参数指定的。

压缩控制变异的平均数量的减少率。标准差线性减小，以便标准差等于 1－Shrink 乘以它在第一代的值。例如，假设 Shrink 缺省值为 1，则变异数在最后一步减小到 0。通过选择绘图函数 Distance(距离)和 Range(范围)能够观察到变异的效果。Rastrigin 函数的遗传算法的运行结果如图 8.43 所示。

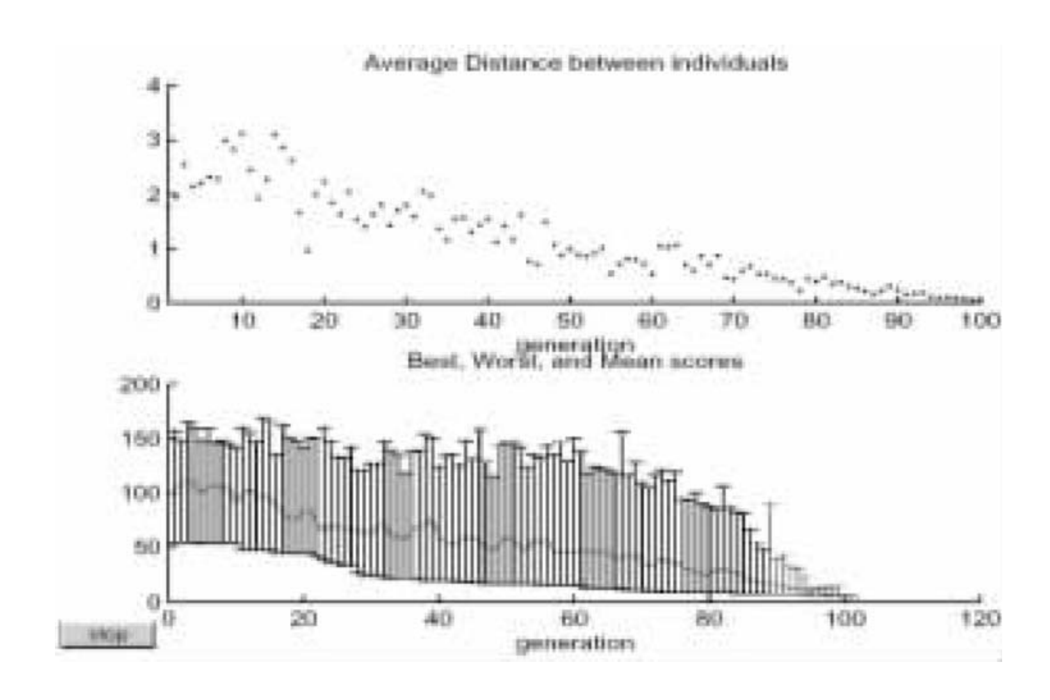



图 8.43　压缩值为 1 时 Rastrigin 函数的距离和范围

图 8.43 的上部图形显示每一代各点之间的平均距离。当变异数减小时，个体之间的平均距离也减小，在最后一代大约减小到 0。图 8.43 下部图形中的垂直线表示每一代适应度值由最小到最大的范围以及适应度值的平均值。当变异数减小时，适应度值的范围也减小。这些图形显示减少变异数，也就减小了子辈的多样性。

作为比较，图 8.44 所示为 Shrink 取 0.5 时的距离和范围的图形。

当 Shrink 设置为 0.5 时，最后一代的平均变异数减小了一半。同样，个体之间的平均距离也大约减小了一半。

**7．** 设置交叉概率

在“Reproduction”选项中，由“Crossover fraction”文本框指定每一种群的一部分，它们不是 Elite children，而是组成的交叉子个体。取交叉概率等于 1，意味着所有子个体都是交叉子个体；取交叉概率等于 0，意味着所有子个体都是变异子个体。下面的例子说明，这两个极端设置都不是有效的函数优化策略。

在这个例子中，定义适应度函数为

$$f(x_1, x_2, \cdots, x_n) = |x_1| + |x_2| + \cdots + |x_n|$$

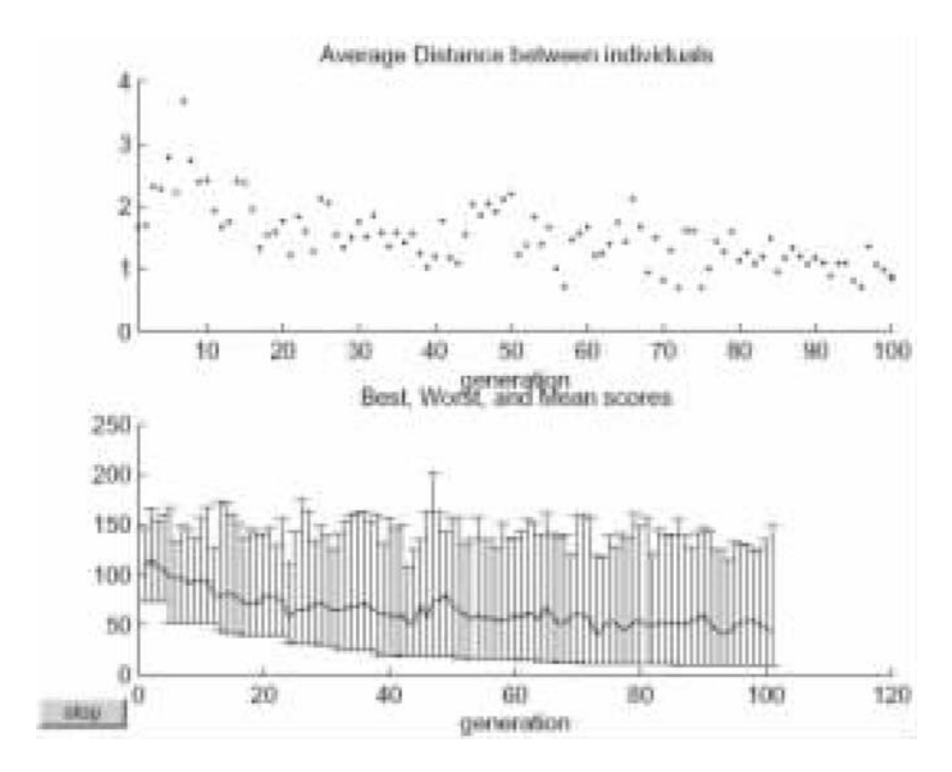



图 8.44　压缩值为 0.5 时 Rastrigin 函数的距离与范围

即点的适应度函数值为所有点的坐标的绝对值之和。

通过设置 Fitness function 为@(x) sum(abs(x))，就可以定义为一个无名函数。

运行这个例子时，有关参数设置如下：

① 设置“Fitness function”为 @(x) sum(abs(x))。

② 设置“Number of variables”为 10。

③ 设置“Initial range”为[−1；1]。

④ 在“Plots”窗格中选择 Best fitness 和 Distance。

首先设置 Crossover fraction 为缺省值 0.8，运行算法，得到最佳适应度值大约为 0.2，如图 8.45 所示。

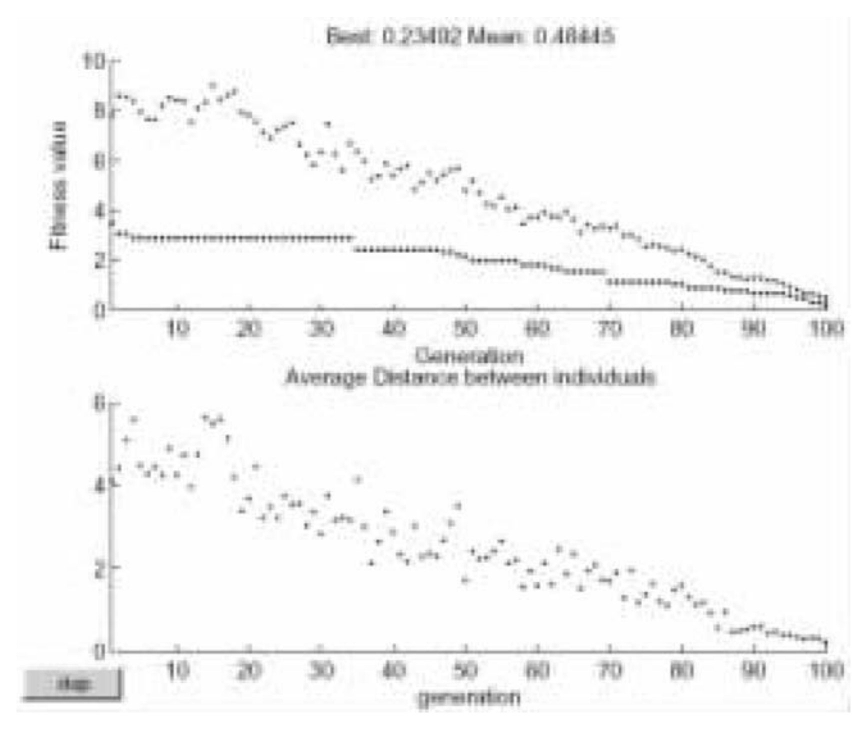



图 8.45　交叉概率为 0.8 时函数的适应度值与平均距离

1）无变异的交叉

为了观察没有变异时遗传算法怎样运行，设置 Crossover fraction 为 1.0，并单击“Start”按钮，得到的最佳适应度值约等于 1.3，如图 8.46 所示。

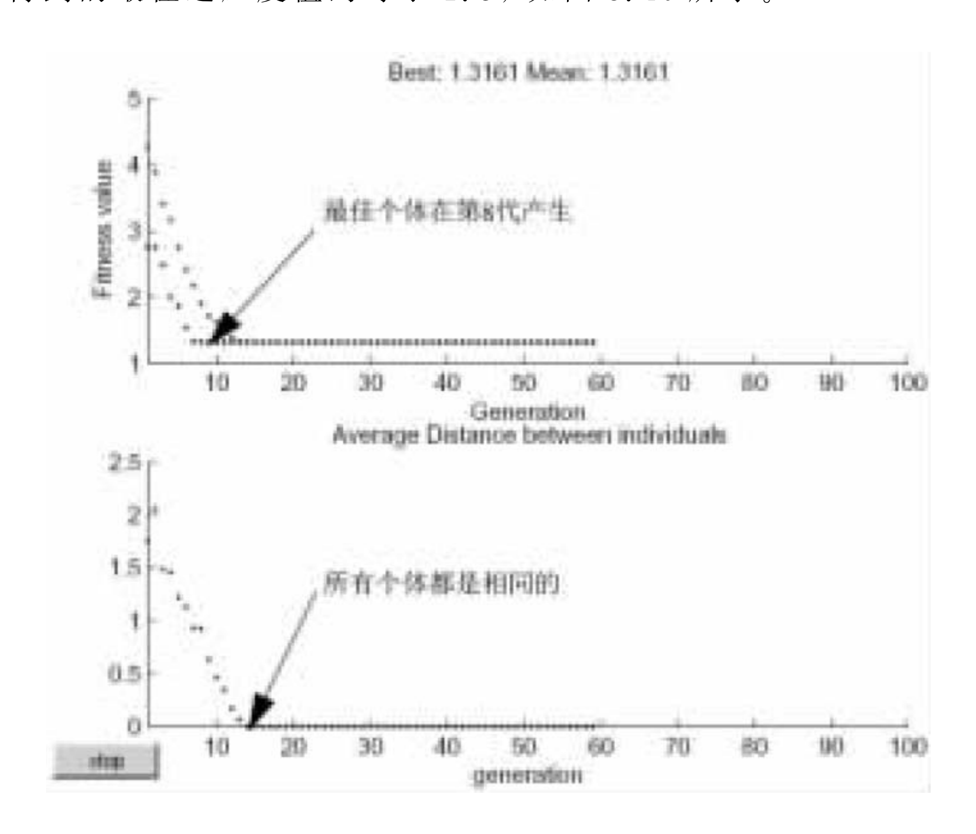



图 8.46　无变异的交叉下函数的适应度值与平均距离

在这种情况下，算法选择初始种群中的个体基因，并把它们重新结合起来。因为没有变异，所以算法不能产生任何新的基因。算法利用这些第 8 代的基因来产生最好的个体，这时最佳适应度图形呈现为水平。此后，它从紧接着的一代选择最佳个体，产生新的最佳个体的副本。到了第 17 代，种群中的所有个体都相同，也就是说，都变成了最佳个体。当这种情形出现时，个体之间的平均距离为 0。算法在第 8 代之后不能改善最佳适应度值，它在 50 代过后就陷于停滞，因为 Stall generations（停滞代数）设置为 50。

2）无交叉的变异

为了查看遗传算法在没有交叉的情形下是如何工作的，设置 Crossover fraction 为 0，并单击“Start”按钮，得到的最佳适应度值约等于 3.5，如图 8.47 所示。

在这种情况下，算法应用的随机变化没有改善第一代最佳个体的适应度函数值，但是，它改善了其他个体的个体基因。由图 8.47 中的上部图形可以看到，适应度函数的平均值逐渐减少，这些改善的基因没有和最佳个体基因结合，因为没有交叉。结果，最佳适应度图形是水平的，并且算法在 50 代停滞。

3）比较遗传算法取不同交叉概率时的运行结果

在工具箱里有一个演示函数 deterministicstudy.m，Crossover fraction 分别设置为 0、0.2、0.4、0.6、0.8、1，把遗传算法应用到 Rastrigin 函数，比较不同的运行结果。遗传算法运行 10 代，对于交叉概率的每一个值，画出每一代之前的所有代的最佳适应度值的均值和标准差。在 MATLAB 命令窗口中输入 deterministicstudy，当演示结束时，画出图形，如图 8.48 所示。

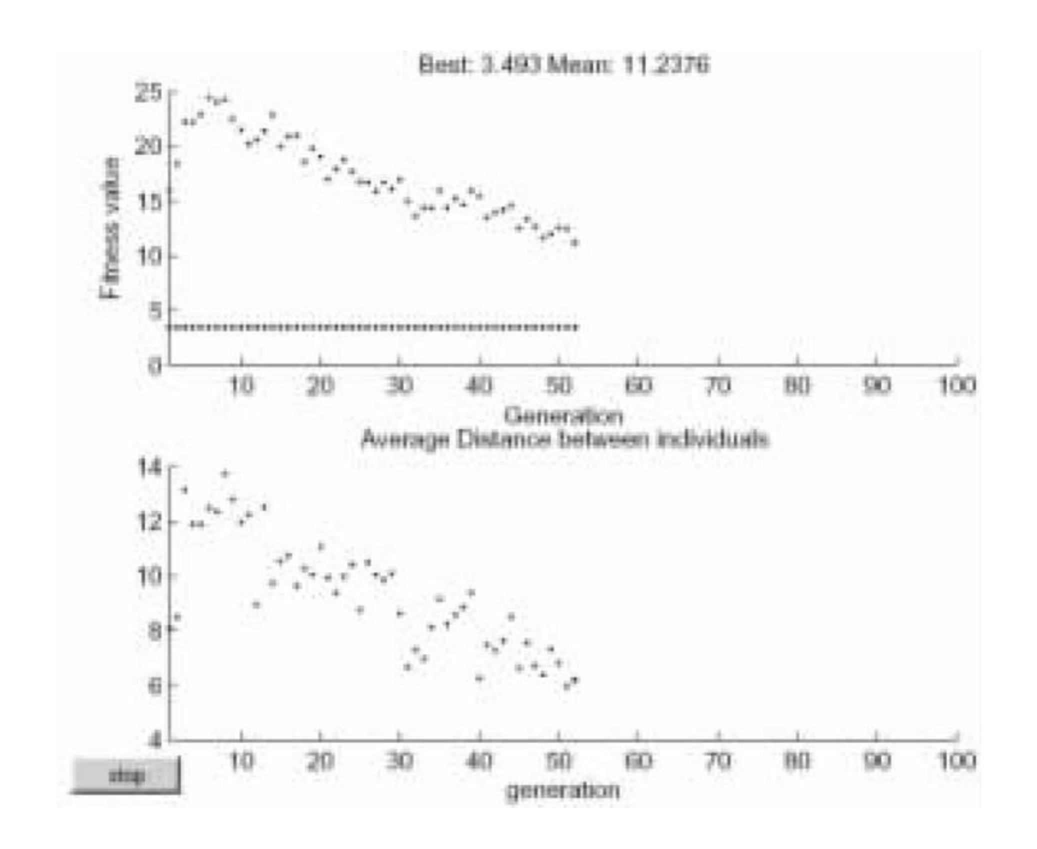



图 8.47　无交叉的变异下函数的适应度值与平均距离

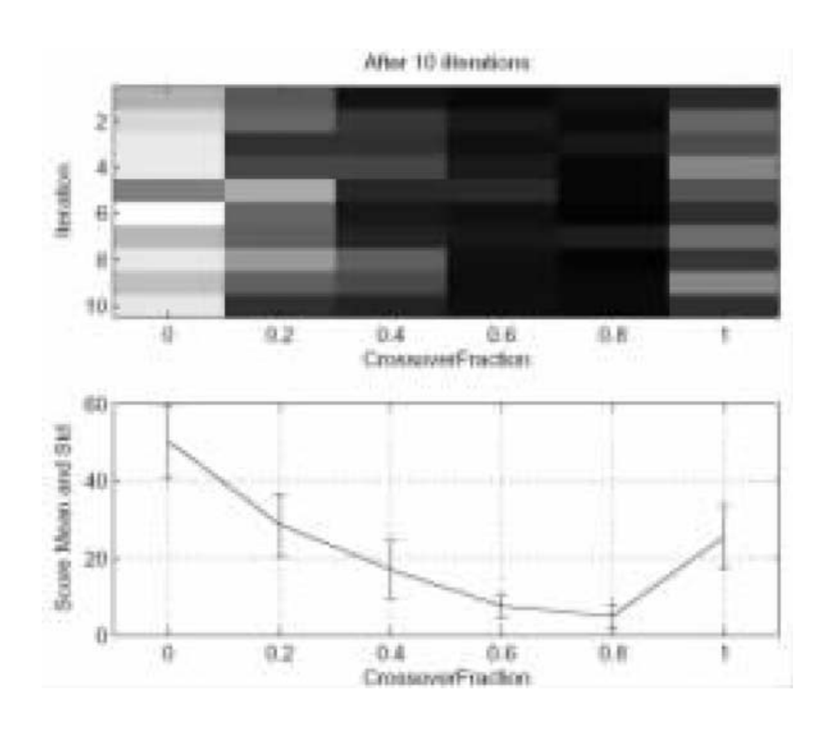



图 8.48　不同交叉概率下最佳适应度值的均值和标准差

图 8.48 中下面的图形显示 10 代不同的交叉概率下最佳适应度值的均值和标准差，上面图形的颜色显示每一代的最佳适应度值。对于这个适应度函数，当 CrossoverFraction 等于 0.8 时得到最好结果。但是，对于其他适应度函数，交叉概率可能取另外的值，才能得到最好结果。

**8.** 举例——相对于全局的局部最小值

有时，优化的目的是要找到函数的全局最小值或最大值——一个点的函数值比搜索空间中其他任何点上的函数值都要小或都要大。但是，最优化算法有时得到的是局部最小值——该点的函数值比它的附近点的函数值小，但是可能比搜索空间的其他点的函数值

大。为了克服这个不足，遗传算法的参数必须设置适当。例如，考虑下面的函数：

$$f(x)=\begin{cases}-\exp\left[-\left(\dfrac{x}{20}\right)^2\right] & x\leqslant 20\\ -\exp(-1)+(x-20)(x-22) & x>20\end{cases}$$

该函数的图形如图 8.49 所示。

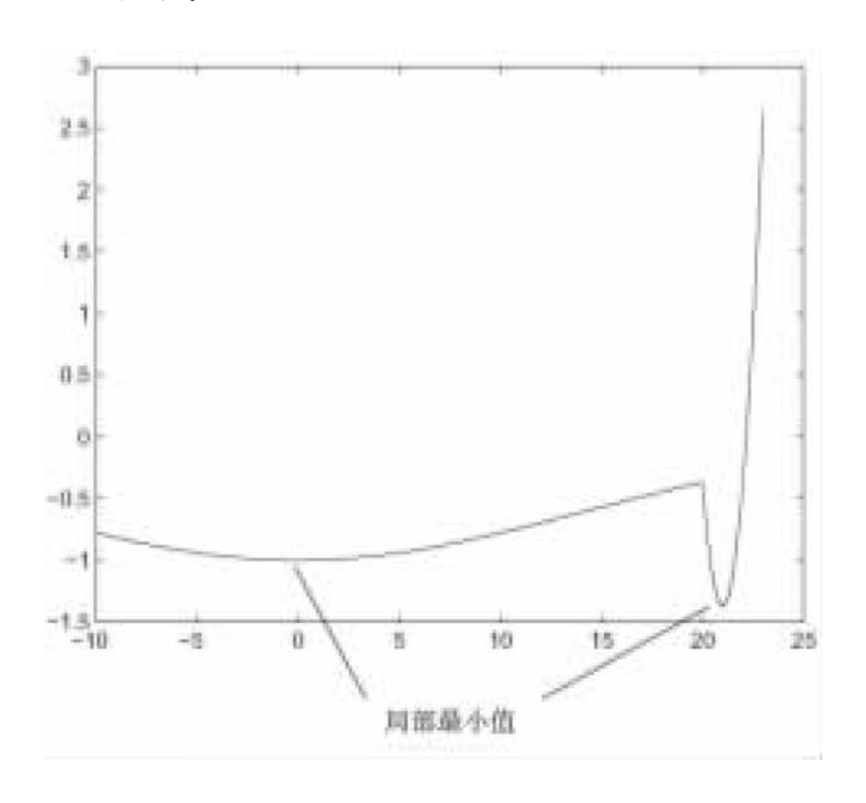



图 8.49 函数 $f(x)$的图形

这个函数有两个局部最小值，当 $x=0$ 时，函数值 $f(x)=-1$；当 $x=21$ 时，函数值 $f(x)=-(1+1/e)\approx-1.36787944117144$，所以，全局最小值是当 $x=21$ 时的函数值。

以下列步骤运行遗传算法：

(1) 用 MATLAB 编辑器将下面的代码复制、粘贴到一个新 M 文件。

```
function y=two_min(x)
if x<20
    y=-exp(-(x/20).^2);
else
    y=-exp(-1)+(x-20)*(x-22);
end
```

(2) 用 MATLAB 将这个文件保存为 two_min.m。

(3) 在遗传算法工具中，设置 Fitness function 为@two_min，设置 Number of variables 为1。

(4) 单击“Start”按钮。

遗传算法返回的值接近局部最小值点 $x=0$，如图 8.50 所示。

图 8.51 说明为什么算法得到的是局部最小值，而不是全局最小值。该图画出了每代个体的范围以及最优个体。

注意：所有个体的范围为$-2\sim2.5$。由于变异，这个范围比缺省 Initial range [0;1]大，但是没有大到搜索全局最小值点 $x=21$ 的附近。

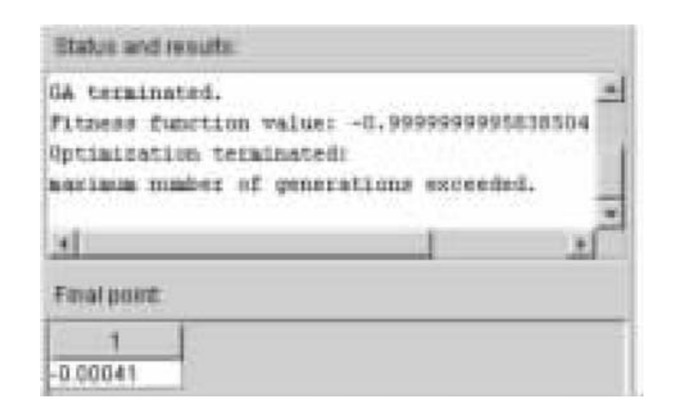



图 8.50　函数 $f(x)$的局部最优解

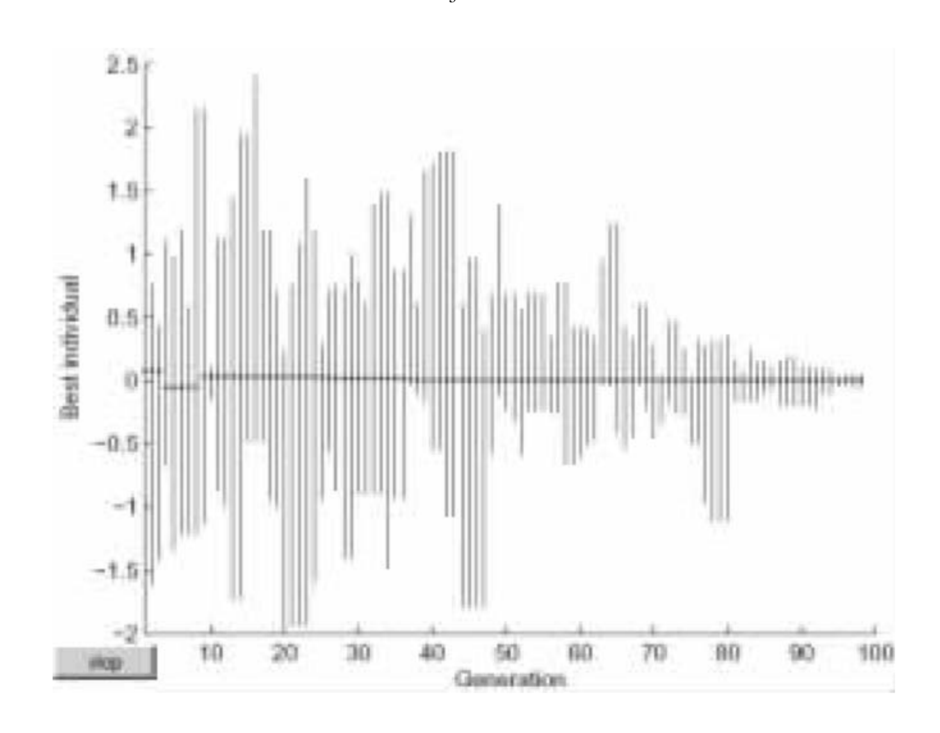



图 8.51　初始范围为[0；1]时每代个体的范围及最优个体

使遗传算法搜索更大范围的点的一个方法是增加种群的多样性，即增大 Initial range。初始范围不一定包含点 $x=21$，但是它必须足够大，以便算法能产生 $x=21$ 附近的个体。设置 Initial range 为[0;15]，如图 8.52 所示。

图 8.52　设置初始范围为[0；15]

单击“Start”按钮，遗传算法返回的值非常接近于 $x=21$ 时的函数值，如图 8.53 所示。

这一次，图形显示个体更大的范围。在第 2 代，有大于 21 的个体，在第 12 代，算法找到大约等于 21 的最优个体，如图 8.54 所示。

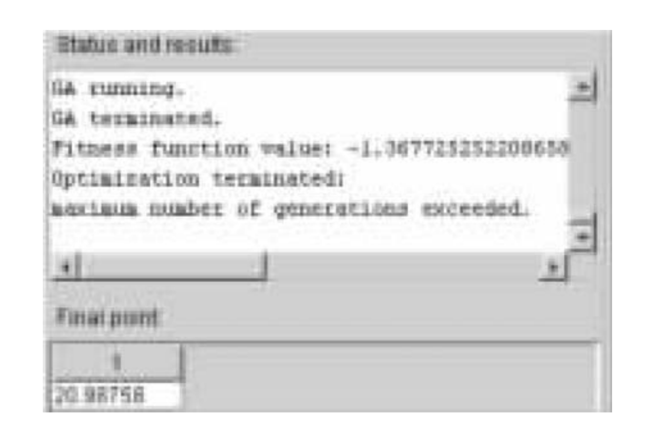



图 8.53　函数 $f(x)$的全部最优解

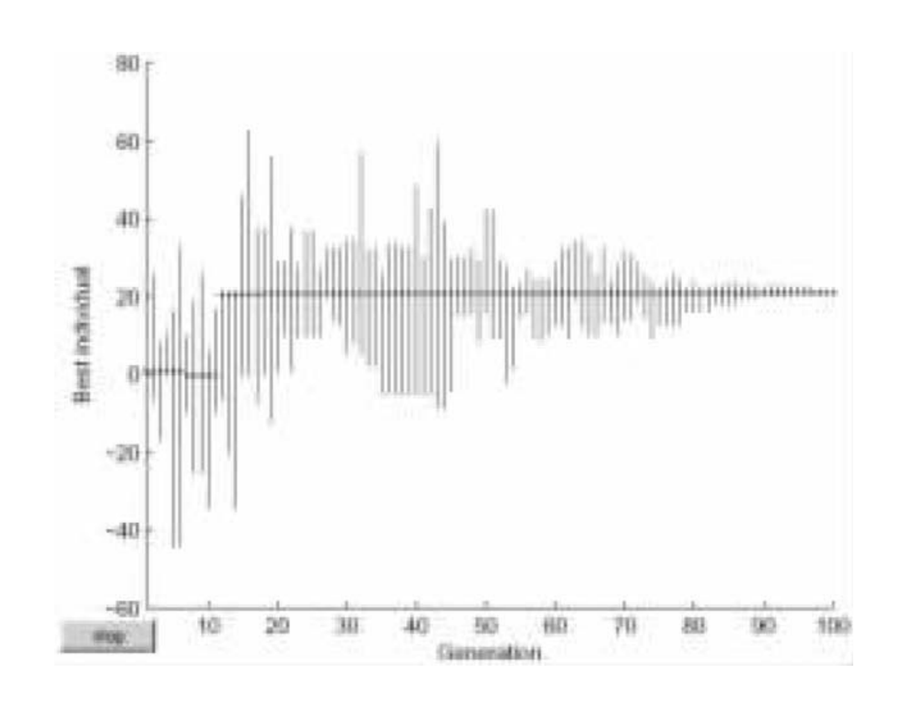



图 8.54　初始范围为[0；15]时每代个体的范围及最优个体

**9**. 使用混合函数

混合函数是一个最优化函数。在遗传算法停止后，为了改善适应度函数值，可以使用混合函数。混合函数将遗传算法得到的最后点作为它的初始点。可以在“Hybrid function（混合函数）”参数域指定混合函数。

作为一个例子，Rosenbrock 函数定义为

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

该函数的图形如图 8.55 所示。

使用最优化工具箱中的一个无约束条件的最小化函数 fminunc 时，首先运行遗传算法，找到最优点附近的一个点，然后，将它作为 fminunc 的初始点，以找到 Rosenbrock 函数的最小值点。

工具箱提供了一个计算该函数值的 M 文件 dejong2fcn.m。为了演示这个例子，需在 MATLAB 编辑窗口提示符下键入 hybriddemo。

为了观察这个例子的运行过程，首先键入 gatool，打开遗传算法工具，进行下列设置：

(1) 设置“Fitness function”为@dejong2fcn。

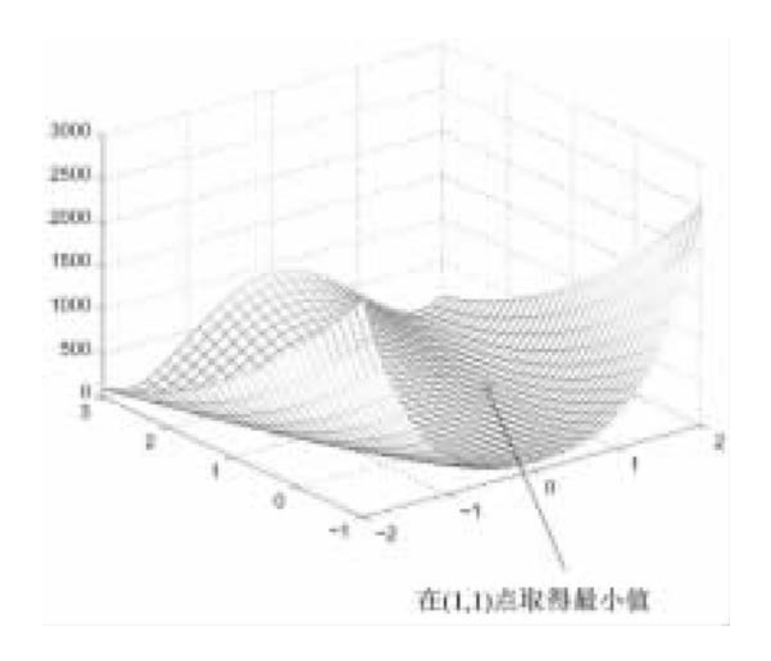



图 8.55 Rosenbrock 函数

(2) 设置“Number of variables”为 2。

(3) 设置“Population size”为 10。

增加混合函数之前，单击“Start”按钮，遗传算法即在“Status and results”窗口中显示出运行结果，如图 8.56 所示。

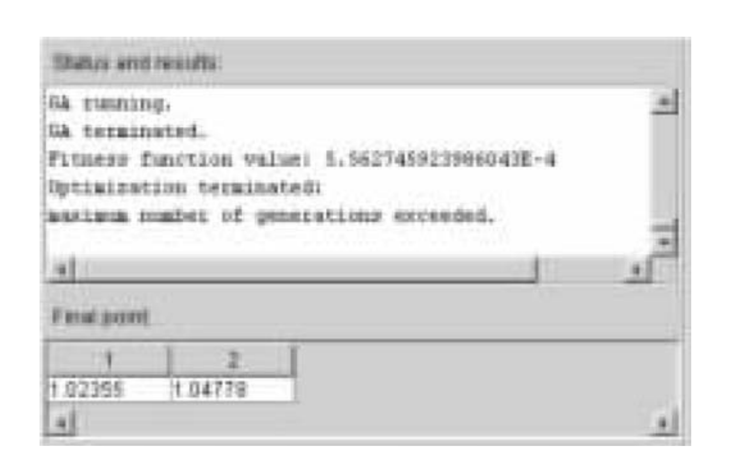



图 8.56 增加混合函数之前的运行结果

最终点的坐标接近问题解的真值(1，1)。在“Hybrid function”参数文本框中设置“Hybrid function”为 fminunc(参见图 8.57)，可以改善这个结果。




图 8.57 设置混合函数为 fminunc

当遗传算法停止时，函数 fminunc 得到遗传算法的最终点，然后将其作为它的初始点，返回更精确的结果，显示在“Status and results”窗口中，如图 8.58 所示。

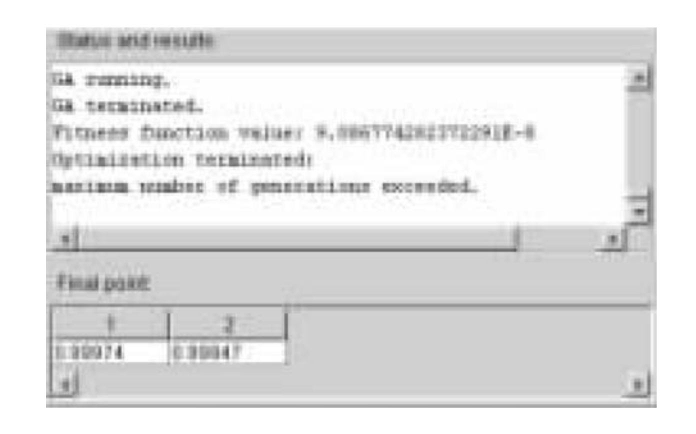



图 8.58　使用混合函数 fminunc 得到的运行结果

**10．**设置最大代数

在“Stopping criteria(停止准则)”中，“Generations(代数)”参数决定最大代数。增加代数，通常可以改善最终结果。例如，改变遗传算法工具中的参数：

(1) 设置“Fitness function”为@rastriginsfcn。

(2) 设置“Number of variables”为 10。

(3) 在“Plots”窗口中选择“Best fitness”。

(4) 设置“Generations”为 Inf。

(5) 设置“Stall generations(停滞代数)”为 Inf。

(6) 设置“Stall time”为 Inf。

然后运行遗传算法大约 300 代，单击“Stop”按钮，运行结果如图 8.59 所示。

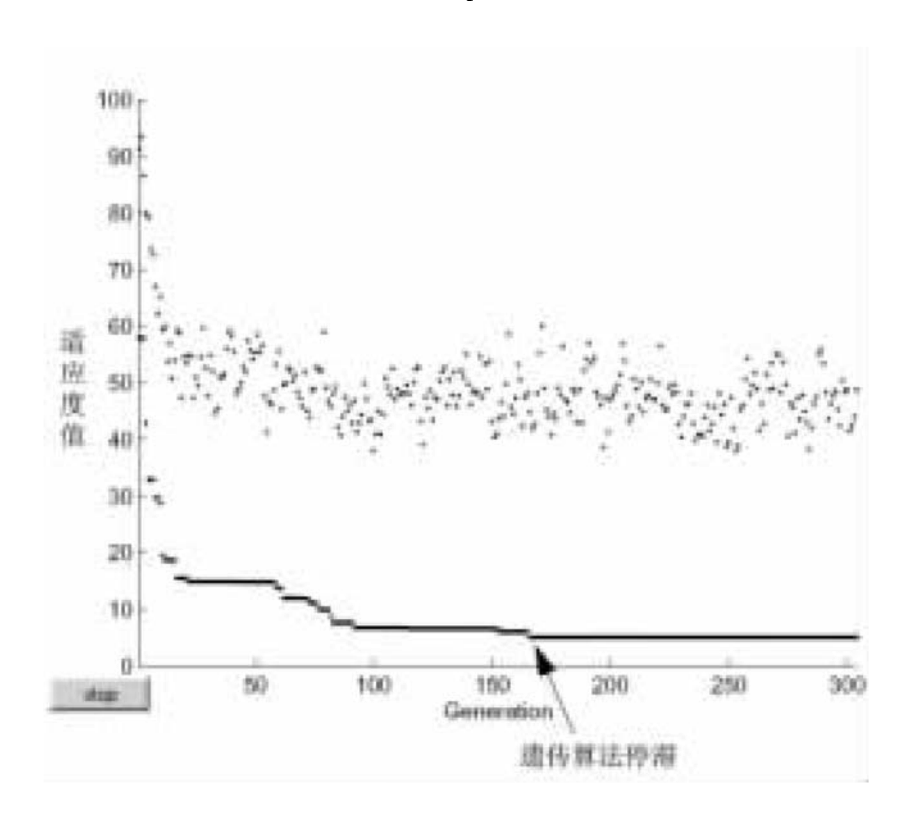



图 8.59　增大代数设置的最佳适应度值

注意：遗传算法在第 170 代时运行停滞—— 170 代后适应度函数值没有明显的变化。如果恢复 Stall generations 为缺省值 50，算法大约在第 230 代停止。如果算法重复利用当前代数的设置停滞，则可以尝试增加 Generations 和 Stall generations 来改善计算结果。然

而，改善其他参数可能效果更好。

另外，需要特别注意的是，当“Mutation function(变异函数)”设置为 Gaussian 时，增加代数的数值，实际得到的最后结果可能更差。这是因为 Gaussian 变异函数有依赖于 Generations 的因素，它可能减小每一代的变异平均数，结果导致 Generations 的设置影响算法的性能。

**11.** 向量化适应度函数

如果向量化适应度函数，则遗传算法一般运行较快。也就是说，遗传算法只调用一次适应度函数，希望适应度函数快速计算当前种群的所有个体的适应度值。

为了向量化适应度函数，首先要编写出计算适应度函数的 M 文件，以便处理具有任意行的一个矩阵，这个矩阵与种群的个体相对应。

例如，适应度函数为

$$f(x_1, x_2) = x_1^2 - 2x_1x_2 + 6x_1 + x_2^2 - 6x_2$$

向量化这个函数，用下面的代码写出一个 M 文件：

```
z=x(:,1).^2 - 2*x(:,1).*x(:,2) + 6*x(:,1) + x(:,2).^2 - 6*x(:,2);
```

其中，x 中的冒号“:”表示 x 的所有行。x(:, 1) 是一个向量。“.^”和“.*”为向量元素之间的运算。

其次，设置“Vectorize(向量化)”参数为 On。

注意：适应度函数必须接受任意行数来使用 Vectorize 参数。

下面是在命令行运行的比较结果，显示出了运行速度的改善情况。在这里我们设置“Vectorize”为 On。

```
tic; ga(@rastriginsfcn,20); toc
elapsed_time =
    4.3660
options=gaoptimset('Vectorize','on');
tic; ga(@rastriginsfcn,20,options); toc
elapsed_time =
    0.5810
```

# 8.4 遗传算法参数和函数

本节详细说明 13 类遗传算法参数和 4 个遗传算法函数，最后给出一个标准算法选项的列表作为总结。

## 8.4.1 遗传算法参数

设置遗传算法参数有两种方法，一种是使用遗传算法工具 GUI，另一种是从命令行调用遗传算法函数 GA。

(1) 如果使用遗传算法工具 GUI，指定参数可从下拉列表中选择一参数或在一文本字段中记入该参数的值。参见 8.3.1 节“使用遗传算法”的“设置选项参数”部分。

(2) 如果从命令行调用函数 GA，指定参数则使用函数 gaoptimset 来创建参数结构，

例如：options＝gaoptimset('param1'，value1，'param2'，value2，...)。参见 8.3.2 节“从命令行使用遗传算法”的“从命令行设置 ga 的参数”部分。

在这一节中，每一个参数用两种方法列出：出现在遗传算法工具中的是标签；出现在参数结构中的是字段名。例如：Population type 在遗传算法工具中作为参数标签；PopulationType 对应参数结构中的字段。

遗传算法参数可划分为以下 13 类：

（1）图形参数。

（2）种群参数。

（3）适应度计算参数。

（4）选择参数。

（5）再生参数。

（6）变异参数。

（7）交叉参数。

（8）迁移参数。

（9）混合函数参数。

（10）停止条件参数。

（11）输出函数参数。

（12）显示到命令窗口参数。

（13）向量参数。

**1. 图形参数**

图形参数工作时可从遗传算法得到图形数据。当选择图形函数并执行遗传算法时，一个图形窗口在分离轴上显示这些图形。可在任意时刻单击图形窗口中的停止按钮来停止这个算法。

“PlotInterval”是指定相邻两次调用图形函数时的遗传代数。

1）绘图参数

在图形方式下可以选择以下任意图形函数：

- Best fitness (@gaplotbestf)——画出最佳函数值与代数对。
- Expectation(期望值)(@gaplotexpectation)——画出与每一代原始得分对应的期望的子代数。
- Score diversity(多样性值)(@gaplotscorediversity)——画出每一代的得分直方图。
- Stopping(停止)(@plotstopping)——画出停止条件水平。
- Best individual(@gaplotbestindiv)——画出每代中最佳适应度个体的向量值。
- Genealogy(家系)(@gaplotgenealogy)——画出个体的谱系。从一代到下一代线条颜色代码如下：红线表示变异的子辈；蓝线表示交叉的子辈；黑线表示原始的个体。
- Scores(@gaplotscores)——画出每一代中个体的得分。
- Distance(@gaplotdistance)——画出每一代中个体间的平均距离。
- Range(@gaplotrange)——画出每一代中最大、最小、平均适应度函数值。
- Selection(@gaplotselection)——画出双亲的直方图。
- Custom function(自定义函数)——能使用自己定义的绘图函数。这个绘图函数只

能在遗传算法工具中使用。选择 Custom function，在文本框中输入@myfun，这里 myfun 是函数名。

当从命令行调用遗传算法函数 GA 来显示图形时，要设置 PlotFcns 字段的参数作为图形函数的句柄。例如，为了显示最佳适应度图形，设置 options(参数)如下：

```
options=gaoptimset('PlotFcns',@gaplotbestf);
```

为了显示多个图形，语法如下：

```
options =gaoptimset('PlotFcns',{@plotfun1,@plotfun2,...});
```

这里@plotfun1，@plotfun2 等是命令行图形函数名。

2）绘图函数的结构

绘图函数的第一行具有如下形式：

```
function state=plotfun(options,state,flag);
```

函数的输入变元是：

- options——包含当前所有 options 设置的结构。
- state——包含当前种群信息的结构。在"状态结构"部分描述了 state 的各个字段。
- flag(字符串)——一个字符串，标志算法的当前运行阶段。

3）状态结构

state 结构是图形函数、变异函数和输出函数的输入参数，包含以下字段：

- Population——当代种群。
- Score——当代种群的得分。
- Generation——当前代数。
- StartTime——GA 的开始时间。
- StopFlag——包含停止原因的字符串。
- Selection——指明被选择出来的优良个体、交叉个体和变异个体。
- Expectation——希望选择的个体数。
- Best——每一代具有最好得分个体的向量。
- LastImprovement——适应度值发生改进的最后一代的代数。
- LastImprovementTime——适应度值发生改进的最后时间。

**2.** 种群参数

种群参数用于确定遗传算法所用种群的参数。

Population type(PopulationType)：指定适应度函数的输入数据类型，可用来设置 Population type 为以下类型之一：

(1) Double Vector(双精度向量)('doubleVector')：当种群中的个体是双精度类型时使用，它是缺省值。

(2) Bit string(位串)('bitstring')：当种群中的个体是位串类型时使用。

(3) Custom(自定义)('custom')：当种群中的个体不是前面两种类型时使用。

(4) 如果使用 Custom ('custom')类型，必须自己编写创建变异和交叉函数来接受这种类型种群输入，并分别在下列域中指定这些函数：

- 创建函数(Creation function )(CreationFcn)。
- 变异函数(Mutation function)(MutationFcn)。

• 交叉函数(Crossover function)(CrossoverFcn)。

Population size (PopulationSize)：指定在每一代中有多少个个体，使用大的种群尺度，遗传算法搜索解空间能更加彻底，同时减少返回局部最小值而不是全局最小值的机会，然而使用大的种群尺度，会使遗传算法运行更慢。

如果设置 Population size 为向量，遗传算法将创建多子种群，子种群的数量等于向量的长度，每个子种群的大小是向量的对应项值。

Creation function(创建函数)(CreationFcn)：指定为 GA 创建初始种群的函数，可选择以下函数：

(1) Uniform (@gacreationuniform)：创建具有均匀分布的随机初始种群，这是缺省值。

(2) Custom：允许使用自己编写的创建函数，使其生成在 Population type 中指定的数据类型。如果使用遗传算法工具，必须指定创建函数：

• 设置 Creation function 为 Custom。
• 设置 Function name(函数名)为@myfun，这里 myfun 是函数名。

如果使用函数 ga，其设置如下：

```
options=gaoptimset('CreationFcn', @myfun);
```

创建函数必须有以下调用语法：

```
function Population=myfun(GenomeLength, FitnessFcn, options)
```

这个函数的输入参数是：

• Genomelength：适应度函数中独立变量的个数。
• FitnessFcn：适应度函数。
• Options：参数结构。

这个函数返回的种群作为遗传算法的初始种群。

Initial population(InitialPopulation)：指定遗传算法的初始种群。缺省值是[ ]，这种情况下 GA 使用 Creation function 创建初始种群；如果输入一个非空数组给 Initial population 域，则这个数组必须是 Population size 行和 Number of variables 列，这种情况遗传算法不调用 Creation function。

种群的 Initial scores(InitialScores)：指定初始种群的初始值。

Initial range(PopInitRange)：指定被创建函数生成的初始种群向量范围，能使用一具有两行、Number of variables 列的矩阵设置 Initial range，每一列具有[lb; ub]形式，这里 lb 是相对项目的下界，而 ub 是上界。如果指定 Initial range 是 2×1 向量，则每个条目均被扩展，行长度不变，即行长度为 Number of variables。

**3．**适应度比例参数

适应度比例参数是把适应度函数返回的适应度值转换为适合选择函数的范围的值。在“Fitness scaling(适应度缩放比例)”窗格中可以指定适应度比例函数参数。

在“Fitness scaling”窗格中，尺度函数选项 Scaling function (FitnessScalingFcn) 是一个下拉列表，可以从中选择要执行适应度比例的函数参数。这些函数参数是：

(1) Rank(排列)(@fitscalingrank)——缺省的适应度比例函数。Rank 函数根据个体适应度值的排列顺序而不是根据个体适应度值的大小来衡量个体的优劣。个体的排列是按

个体适应度值排序的。最适应个体的排序为1，次最适应个体的排序为2，依次类推。Rank函数按适应度比例进行排序，从而消除了原始适应度值的影响。

（2）Proportional（比率）（@fitscalingprop）——比率的计算使个体的适应度比例大小与它的适应度值成比例。

（3）Top（最佳）（@fitscalingtop）——计算最佳比例等同于计算最佳个体。选择这一项后，最佳个体比例显示在另外一个字段“Quantity（数量）”中。“Quantity”规定了指派正的比例值的个体数目。“Quantity”可以是1到种群大小之间的整数，也可以是0到1之间的小数，这个小数是种群大小的百分比，其缺省值是0.4。每个能产生子辈的个体指派给相同的比例值，而其他个体的比例值指派为0。这个比例值的形式为：[0 1/n 1/n 0 0 1/n 0 0 1/n ...]。

在命令行改变“Quantity”的缺省值，可使用如下语句：

```
options=gaoptimset('FitnessScalingFcn',{@fitscalingtop,quantity})
```

这里，quantity是“Quantity”的值。

（4）Shift linear（线性转换）（@fitscalingshiftlinear）——利用线性转换来衡量适应度值，将使最适应个体的期望值等于个体的平均值乘以一个常数。在“Max survival rate（最大生存率）”字段中，可以设置这个常数。如果选择线性转换，这个字段显示的缺省值是2。

在命令行改变“Max survival rate”的缺省值，可使用下面的语句：

```
options=gaoptimset('FitnessScalingFcn',{@fitscalingshiftlinear,rate})
```

这里，rate是“Max survival rate”的值。

（5）Custom（定制）——能够使用户编写自己的尺度函数。可以使用遗传算法工具来指定尺度函数：

- 设置“Scaling function（尺度函数）”为Custom。
- 设置“Function name（函数名）”为@myfun，这里myfun是函数名。

如果在命令行使用ga，可使用下面的语句：

```
options=gaoptimset('FitnessScalingFcn',@myfun);
```

尺度函数必须遵循下面的语法格式：

```
function expection=myfun(scores,nParents)
```

这个尺度函数的输入参数是：

- scores——标量向量，作为种群的成员。
- nParents——这个种群所必需的父辈个体数目。

这个函数返回的期望值expectation是一个与scores长度相同的标量行向量，给出种群中每个成员的尺度值。expectation的总项数必须等于nParents。

**4. 选择参数**

选择参数规定遗传算法怎样为下一代挑选双亲，可用以下方法指定算法函数：

Selection function（SelectionFcn）域在面板的Selection参数内，这些参数是：

（1）Stochastic uniform（随机均匀分布）（@selectionstochunif）——缺省的选择函数为Stochastic uniform函数，该函数布局在一条线上，每一父辈根据其刻度值按比率对应线上的一部分，算法以相同大小的步长沿线移动。在每一步，算法根据降落的位置确定一父辈，第一步是一小于步长的均匀随机值。

(2) Remainder(剩余)(@selectionremainder)——剩余选择分配其双亲由每个个体刻度值的整数部分决定，并随后在剩余的小数部分采用轮盘赌选择方法。例如，一个个体的刻度值是 2.3，由于其整数部分是 2，这个个体在父辈表中出现两次。随后这些父辈按刻度值的整数部分进行分配，其余的父辈随机选出。父辈在这一步被选取的概率是刻度值的小数部分。

(3) Uniform(均匀)(@selectionuniform)——均匀选项用父辈的代数和期望值来选择父辈，均匀选择用于调试和测试，但不是一个非常有效的搜索策略。

(4) Roulette(轮盘赌选项)(@selectionroulette) ——轮盘赌选项挑选父辈时使用一个模拟的轮盘赌，个体在轮子上所占的区域与个体的期望值成正比，算法使用一个随机数选择一个概率与其相等的区域。

(5) Tournament(锦标赛选项)(@selectiontournament)——锦标赛选项通过挑选"Tournament size(锦标赛大小)"随机数生成器挑选每一个父个体，随后选择它们中缺少的最好的个体加入父辈中，"Tournament size"至少为 2，缺省值是 4。

在命令行用以下语法来改变缺省值：

```
options=gaoptimset('SelectionFcn',{@selecttournament,size})
```

这里"size"是"Tournament size"的值。

(6) Custom——允许编写自己的选择函数，在遗传算法工具中为了指定这个函数，必须

- 设置 Selection function(选择函数)为"Custom"。
- 设置"Function name"为@myfun，这里 myfun 是函数名。

如果使用命令行，可输入以下语句：

```
options=gaoptimset('SelectionFcn',@myfun)
```

选择函数必须具有以下调用语法：

```
function parents=myfun(expectation,nParents,options)
```

这个函数的输入参数是：

- expectation(期望值)——期望的子辈个体数量作为种群的成员。
- nParents(父辈个体)——选择的父辈个体的数量。
- options——遗传算法参数结构。

这个函数返回父辈，它是具有 nParents 长且包含选择的父辈个体指示的行向量。

**5.** 再生参数

再生参数说明了遗传算法怎样为下一代创建子个体。

Elite count(EliteCount)——指定将生存到下一代的个体数，设置 Elite count 为一个小于或等于种群尺度的正整数，其缺省值是 2。

Crossover fraction(CrossoverFraction)——指定下一代中不同于原种群的部分，它们由交叉产生。Crossover fraction 是一个 0 到 1 之间的小数，可在文本框中输入或移动滑槽进行设置，其缺省值是 0.8。

可参见 8.3.3 节"设置交叉概率"中的例子。

**6.** 变异参数

变异参数说明遗传算法怎样通过小的随机数改变种群中的个体而创建变异的子辈。

变异提供遗传变异功能而使遗传算法搜索更广泛的空间。在“Mutation(变异)”面板的“Mutation function(变异函数)”(MutationFcn)字段来指定变异函数，可选择以下函数：

(1) Gaussian(高斯函数)(mutationgaussian)——这是缺省的变异函数，把一高斯分布、具有均值0的随机数加到父向量的每一项。这个分布的变化由参数“Scale”和“Shrink”决定。如果选择Gaussian，它们将显示出来，且在“Population”参数中按照“Initial range”进行设置。

① Scale参数确定第一代的方差，如果设置“Initial range”为1行2列的向量v，其初始方差同所有父向量坐标相同，且由Scale * (v(2) − v(1))给出。如果设置“Initial range”为2行、Number of variables列的向量，父向量坐标为i的初始方差由Scale * (v(i，2)−v(i，1))给出。

② Shrink参数控制方差怎样随着迭代次数增大而收缩。如果设置“Initial range”为1行2列的向量，则第$k$代的方差$\mathrm{var}_k$与父向量坐标相同，且由下列公式给出：

$$\mathrm{var}_k=\mathrm{var}_{k-1}\left(1-\mathrm{shrink}\cdot\frac{k}{\mathrm{generations}}\right)$$

如果设置“Initial range”为2行、Number of variables列的向量，对父向量坐标为$i$的初始方差由下列公式给出：

$$\mathrm{var}_{i,\,k}=\mathrm{var}_{i,\,k-1}\left(1-\mathrm{shrink}\cdot\frac{k}{\mathrm{generations}}\right)$$

如果设置“Shrink”为1，则算法压缩方差接近线性坐标直到最后一代达到0，一个负的“Shrink”引起方差的增长。缺省的“Scale”和“Shrink”分别是0.5和0.7。在命令行要改变缺省值，可使用如下语法：

```
options=gaoptimset('MutationFcn', ...{@mutationgaussian, scale, shrink})
```

这里scale和shrink分别是“Scale”和“Shrink”的值。

(2) Uniform (mutationuniform)——均匀变异是两个过程。第一步，算法选择个体变量的一部分进行变异，这里每一项有一Mutation Rate(变异概率)，这个Rate的缺省值是0.01；第二步，算法用均匀选择在项目范围中选择一随机数替换每个选中的项目。在命令行要改变Rate的缺省值，可使用如下语法：

```
options=gaoptimset('MutationFcn', {@mutationuniform, rate})
```

这里rate是Rate的值。

(3) Custom——自定义允许使用自己的变异函数，使用遗传算法工具时，指定变异函数使用如下：

- 设置“Mutation function”为Custom。
- 设置“Function name”为@myfun，这里myfun是自己设计的变异函数名。

如果使用ga函数，则用语句options=gaoptimset('MutationFcn', @myfun)。自定义的变异函数必须有如下调用格式：

```
function mutationChildren=myfun(parents, options, nvars,FitnessFcn, ...
    state, thisScore, thisPopulation)
```

这个函数的自变量是：

- parents——被选择函数选择出的父辈的行向量。

- options——参数结构。
- nvars——变量数。
- FitnessFcn——适应度函数。
- state——包含当前种群信息的结构(在本节"状态结构"部分中描述了 state 域)。
- thisScore——许多当前种群的向量。
- thisPopulation——当前种群的个体矩阵。

这个函数返回 mutationChildren(变异的子辈),它就像一个矩阵,其行对应子孙,矩阵的列数就是"Number of variables"。

**7. 交叉参数**

交叉参数说明遗传算法如何组合两个个体或双亲,为下一代形成一交叉的子个体。

Crossover function(CrossoverFcn):指明进行交叉的函数,可以选择以下函数:

(1) Scattered(分散)(@crossoverscattered):这是一个缺省的交叉函数,它创建一个二进制向量,如果这个向量某位是 1,则这个基因从第一个父辈中来,如果为 0,则从第二个父辈中来,组合这些基因而形成一子个体。例如:父辈 p1,p2 是:

p1=[a b c d e f g h]

p2=[1 2 3 4 5 6 7 8]

这个二进制向量是[1 1 0 0 1 0 0 0],则函数返回如下子辈个体:

child1=[a b 3 4 e 6 7 8]

(2) Single point (@crossoversinglepoint)——单点交叉,它在 1 到"Number of variables"之间选择一随机数 n,随后:

- 在第一个父辈中选择序号小于或等于 n 的向量项。
- 在第二个父辈中选择序号大于 n 的向量项。
- 连接这些项目形成一子辈。

例如,父辈 p1,p2 是:

p1=[a b c d e f g h]

p2=[1 2 3 4 5 6 7 8]

且交叉点是 3,则函数返回下列子辈:

child=[a b c 4 5 6 7 8]

(3) Two point (@crossovertwopoint)——两点交叉,它在 1 到"Number of variables"之间选择两个随机数 m 和 n,随后:

- 在第一个父辈中选择序号小于或等于 m 的向量项。
- 在第二个父辈中选择序号为 m+1～n 的向量项。
- 序号大于 n 的向量项也来自于第一个父辈。

算法连接这些基因形成单一基因,例如父辈 p1,p2 是:

p1=[a b c d e f g h]

p2=[1 2 3 4 5 6 7 8]

且交叉点是 3 和 6,则函数返回的子辈是

child=[a b c 4 5 6 g h]

(4) Intermediate (@crossoverintermediate)——通过父辈的加权平均值创建子辈,可

通过一简单参数“Ratio(比率)”指定权值，“Ratio”可能是一标量或具有“Number of variables”长的行向量，缺省值是向量的每个值均为1，这个函数使用下列公式从双亲计算出子辈：

child＝parent1 ＋ rand ＊ Ratio ＊ ( parent2 － parent1)

如果“Ratio”的所有项值均在[0，1]范围内，产生的子辈限于由父母的相对顶点定义的立体空间中，如果“Ratio”不在这个范围，则子辈可能位于这个空间之外。如果“Ratio”是一个标量，则所有子辈都将位于父母间的一直线上。在命令行方式下改变“Ratio”的缺省值，可使用以下语法：

options＝gaoptimset('CrossoverFcn'，...{@crossoverintermediate，ratio})

在这里，ratio是“Ratio”的值。

(5) Heuristic (@crossoverheuristic)——返回的子辈位于包含父辈的直线上，离父辈不远的距离上有较好的适应度；在同一方向上远离父辈，则有较差的适应度。可以使用参数“Ratio”指定子辈离较好适应度的父辈有多远，这个参数是在选择启发式(Heuristic)时出现的。“Ratio”的缺省值是1.2，如果父辈是parent1和parent2，而parent1有较好的适应度，这个函数返回的子辈如下：

child＝parent2 ＋ Ratio ＊ (parent1 － parent2)

在命令行改变“Ratio”的缺省值，使用如下语法：

options＝gaoptimset('CrossoverFcn',{@crossoverheuristic，ratio})

在这里，ratio是“Ratio”的值。

(6) Custom——允许使用自己定义的交叉函数。在遗传算法工具中，可使用如下方法指定交叉函数：

- 设置“Crossover function”为Custom。
- 设置“Function name”为@myfun，这里myfun是自定义的函数名。

如果使用GA，则设置options＝gaoptimset('CrossoverFcn'，@myfun)。

自定义函数必须有如下调用格式：

xoverKids＝myfun(parents，options，nvars，FitnessFcn，unused，thisPopulation)

在这里，函数的自变量是：

- parents——通过选择函数来选择双亲的行向量。
- options——参数结构。
- nvars——Number of variables(基因数)。
- FitnessFcn——适应度函数。
- unused——保留，未使用。
- thisPopulation——表示当前种群的矩阵。这个矩阵的行数是“Population size(种群尺度)”，列数是“Number of variables(变量个数)”。

这个函数返回xoverKids(即交叉的子辈)，它是一个矩阵，每行对应子辈，其列数是“Number of variables”。

**8.** 迁移参数

Migration(迁移)指明个体在子种群间怎样移动，如果设置“Population size”为一长度大于1的向量，则迁移发生。当迁移发生时，一个子种群中最好的个体代替另一子种群中

最差的个体。个体从一个子种群迁移到另一子种群是复制，即在源子种群中并没有被移走。

可以通过“Migration”参数面板中下面三个字段来控制迁移的发生：

(1)“Direction(方向)”(MigrationDirection)——迁移发生在一个或两个方向。

如果设置“Direction”为 Forward ('forward')，则迁移发生在下一个种群，也就是第 N 个子种群迁移到第 N+1 个子种群。

如果设置“Direction”为 Both ('both')，则第 N 个子种群迁移到第 N−1 个子种群和 N+1 个子种群。

迁移在最后一个子种群处将卷绕回来，即最后一个子种群迁移到第一个子种群，第一个子种群可以迁移到最后一个子种群。为了防止卷绕，在确定的种群尺度下，在种群尺度向量的最后添加一 0 项，指示一大小为 0 的子种群。

(2)“Interval(间隔)”(MigrationInterval)——指明在两次迁移间要经过多少代，例如设置 Interval 为 20，则每隔 20 代就发生迁移。

(3)“Fraction(百分比)”(MigrationFraction)——指明在两个子种群间有多少个个体迁移。“百分比”指明两个子种群中较小子种群的个体迁移百分比。例如：如果个体从一个有 50 个个体的子种群迁移到 1 个有 100 个个体的子种群，且 Fraction 设置为 0.1，则发生迁移的个体数是 0.1×50=5。

**9. 混合函数参数**

混合函数是运行在遗传算法终止后的另一个最小化函数，可在“Hybrid function(混合函数)”(HybridFcn)参数中指定混合函数。混合函数有以下选择：

(1) [ ]——没有混合函数。

(2) fminsearch (@fminsearch)——使用 MATLAB 函数 fminsearch。

(3) patternsearch (@patternsearch)——使用模式搜索。

(4) fminunc (@fminunc)——使用优化工具箱函数 fminunc。

**10. 停止条件参数**

停止条件决定什么引起算法的终止，可以指明以下参数：

(1)“Generations”——指明算法最大重复执行次数，缺省值是 100。

(2)“Time limit”(TimeLimit)——指明算法停止执行前的最大时间，以秒为单位。

(3)“Fitness limit(适应度限)”(FitnessLimit)——最好适应度值小于或等于“Fitness limit”，则算法终止。

(4)“Stall generations(停滞代数)”(StallGenLimit)——如果适应度值在“Stall generations”指明的代数没有改进，则算法停止。

(5)“Stall time(停滞时间)”(StallTimeLimit)——如果最好适应度值在“Stall time”时间间隔内没有改进，则算法终止。

**11. 输出函数参数**

输出函数在每一代返回来自遗传算法的输出到命令行。

“History to new window(记录到新窗口)”(@gaoutputgen)——每到“Interval”的倍数，重复在新窗口显示由算法计算的历史点。

“Custom”——允许使用自己的输出函数，在遗传算法工具中使用以下方法指明输出

函数：

- 选择“Custom function(自定义函数)”。
- 在文本框输入@myfun，这里 myfun 是函数名。

如果使用 GA，则设置 options＝gaoptimset('OutputFcn'，@myfun)。

在 MATLAB 命令行键入

```
edit gaoutputfcntemplate
```

则可使用模板编写自己的输出函数。

输出函数有如下调用语法：

```
[state，options，optchanged]＝myfun(options，state，flag，interval)
```

函数有如下输入参数：

- options——参数结构。
- state——包含当前代信息的结构。
- flag——指明算法当前状态的字符串，有如下形式：'init'表示初始状态；'iter'表示算法运行状态；'done'表示算法终止状态。
- interval——可选的 interval 自变量。

输出函数返回如下结果变量：

- state——包含当前代信息的结构。
- options——被输出函数修改的参数结构，这个参数是可选的。
- optchanged——指示 options 是否改变的标志。

**12.** 显示到命令窗口参数

“Display(显示级别)”——指明遗传算法运行时，在命令行显示的信息数，有效的参数是：

- Off ('off')——只有最终结果显示。
- Iterative ('iter')——显示每一次迭代的有关信息。
- Diagnose ('diagnose')——显示每一次迭代的信息，而且还列出函数缺省值已经被改变的有关信息。
- Final ('final')——遗传算法的结果(成功与不成功)、停止的原因、最终点。

Iterative 与 Diagnose 两者显示如下信息：

- Generation——代数。
- f-count——适应度函数估价的累计数。
- Best f(x)——最佳适应度值。
- Mean f(x)——平均适应度值。
- Stall Generations——停滞代数，即最后一次改进适应度函数以来的代数。

“Display”的缺省值是：

- Off——遗传算法工具中。
- 'final'——使用 gaoptimset 创建的参数结构中。

**13.** 向量参数

“向量参数”指明适应度函数的计算是否被向量化。如果设置“Fitness function is vectorized(适应度函数向量化)”(Vectorized)为 Off，则遗传算法在每次循环中计算新一代

个体的适应度值；如果设置“Fitness function is vectorized”为 On，则遗传算法计算新一代个体的适应度值时只调用适应度函数一次。无论如何，使用这个参数，适应度函数必须能接受具有任意行数的输入矩阵。

### 8.4.2 遗传算法函数

本节介绍遗传算法工具的四个函数的功能、格式及其详细说明，这四个函数分别为：ga、gaoptimget、gaoptimset 和 gatool。

**1．函数 ga**

功能：用遗传算法搜索函数最小值。

格式：

```
x=ga(fitnessfun，nvars)
x=ga(fitnessfun，nvars，options)
x=ga(problem)
[x，fval]=ga(...)
[x，fval，reason]=ga(...)
[x，fval，reason，output]=ga(...)
[x，fval，reason，output，population]=ga(...)
[x，fval，reason，output，population，scores]=ga(...)
```

详细说明：

x=ga(fitnessfun，nvars)：把遗传算法应用到优化问题，这里 fitnessfun 是最小化的目标函数，nvars 是找到的最优个体答案向量 x 的长度。

x=ga(fitnessfun，nvars，options)：把遗传算法应用到优化问题，使用参数结构中的那些选项参数。

x=ga(problem)：为 problem 找最小值，problem 结构有下列三个字段：

- fitnessfcn——适应度函数。
- nvars——适应度函数的独立变量个数。
- options——用 gaoptimset 创建的参数结构。

[x，fval]=ga(...)：返回 fval，即适应度函数在 x 处的值。

[x，fval，reason]=ga(...)：返回 reason，它是一包含算法终止原因的字符串。

[x，fval，reason，output]=ga(...)：返回 output，它是一包含每一代输出和算法执行的其他信息的结构，这个输出结构包含以下字段：

- randstate——遗传算法启动之前 rand 的状态，rand 是 MATLAB 随机数生成器。
- randnstate——遗传算法启动之前 randn 的状态，randn 是 MATLAB 普通随机数生成器，可使用 randstate 和 randnstate 值再现 GA 的输出。
- generations——计算的代数。
- funccount——适应度函数的估算次数。
- message——算法终止的原因，它与输出变量 reason 相同。

[x，fval，reason，output，population]=ga(...)：返回 population，它是种群矩阵，rows 是最后的种群。

[x，fval，reason，output，population，scores]=ga(...)：返回值 scores，它是最终种群的得分值。

注意：对 problem 使用的种群类型是双精度型向量，ga 并不接受输入是复数型向量的函数，为了解决包括复数数据的问题，可以编写自己的函数，即把复数的实部和虚部分开，变为 ga 能接受的实数向量。

举例：

```
[x fval，reason]=ga(@rastriginsFcn，10)
x=
    Columns 1 through 7
    0.9977 0.9598 0.0085 0.0097 -0.0274 -0.0173 0.9650
    Columns 8 through 10
    -0.0021 -0.0210 0.0065
fval=
    3.7456
reason=
    generations
```

参见：gaoptimset，gatool。

**2．函数 gaoptimget**

功能：得到遗传算法参数结构值。

格式：

```
val=gaoptimget(options，'name')
```

详细说明：

val=gaoptimget(options，'name')：返回参数 name 的值，name 来自遗传算法参数的结构参数。

如果 name 的值没有在参数中指明，则 gaoptimget(options，'name')返回一空的矩阵。它只需要能惟一定义 name 足够的前导字符。gaoptimget 忽略参数 name 中的大小写。

参见：ga，gatool。

**3．函数 gaoptimset**

功能：创建遗传算法参数结构。

格式：

```
options=gaoptimset
gaoptimset
options=gaoptimset('param1',value1,'param2',value2,...)
options=gaoptimset(oldopts,'param1',value1,...)
options=gaoptimset(oldopts,newopts)
```

详细说明：

options=gaoptimset (无输入参数)：创建一称为 options 的选项参数结构(它包含遗传算法的参数)，并设置这些参数为缺省值。

gaoptimset：无输入或输出变量参数，显示具有有效值的参数的完整列表。

options＝gaoptimset('param1',value1,'param2',value2,...)：创建一结构 options，并设置其他值，即'param1'为 value1，'param2'为 value2，等等。在这里，只需要给出能惟一定义参数名的足够的前导字符，任何未指定的参数设置均使用它们的缺省值。参数名中的大小写被忽略。

options＝gaoptimset(oldopts，'param1'，value1,...)：创建一 oldopts 拷贝，修改指定的参数具有指定的值。

options＝gaoptimset(oldopts，newopts)：用一新的参数结构 newopts 组合一存在的参数结构 oldopts，在 newopts 中任意参数的非空值覆盖 oldopts 中对应的参数。

表 8.2 列出了能用 gaoptimset 设置的参数，在 8.4.1 节“遗传算法参数”中完整地描述了这些参数及其取值。{}中的值是缺省值。在命令行使用无输入参数的命令函数 gaoptimset，可查看这些优化参数。

表 **8.2** 使用函数 **gaoptimset** 设置的参数

| 参　　数 | 说　　明 | 值 |
|---|---|---|
| CreationFcn | 创建初始种群的函数句柄 | {@gacreationuniform} |
| CrossoverFraction | 不包括优良子辈，由交叉函数创建的下一代种群的交叉概率 | 正数标量 \| {0.8} |
| CrossoverFcn | 遗传算法用来创建交叉的子辈函数的句柄 | @crossoverheuristic<br>{@crossoverscattered}<br>@crossoverintermediate<br>@crossoversinglepoint<br>@crossovertwopoint |
| EliteCount | 正整数，指明当前代中有多少个个体一定生存到下一代 | 正整数 \| {2} |
| FitnessLimit | 标量，如果适应度值达到 FitnessLimit 的值，则遗传算法停止 | 标量 \| {－Inf} |
| FitnessScalingFcn | 变换适应度函数值的函数的句柄 | @fitscalinggoldberg<br>{@fitscalingrank}<br>@fitscalingprop<br>@fitscalingtop |
| Generations | 正整数，指明算法停止前可迭代的最大次数 | 正整数\| {100} |
| PopInitRange | 矩阵或向量，指明初始种群中个体的范围 | 矩阵或向量[0;1] |
| PopulationType | 字符串，指定种群的数据类型 | 'bitstring' \| 'custom' \| {'doubleVector'} |
| HybridFcn | 在 ga 终止后使用它继续优化的函数的句柄 | 函数句柄 \| {[ ]} |
| InitialPopulation | 初始种群 | 正标量 {[ ]} |

| 参　　数 | 说　　明 | 值 |
| --- | --- | --- |
| InitialScores | 初始得分 | 列向量 \| {[ ]} |
| MigrationDirection | 迁移方向 | 'both' \| {'forward'} |
| MigrationFraction | 0 到 1 之间的标量，指明每个子种群迁移到不同子种群的个体比率 | 标量 \| {0.2} |
| MigrationInterval | 正整数，指明间隔多少代子种群间个体发生迁移 | 正整数 \| {20} |
| MutationFcn | 产生变异子辈的函数的句柄 | @mutationuniform<br>{@mutationgaussian} |
| OutputFcns | ga 在每次迭代调用的函数的句柄数组 | 数组 \| {[ ]} |
| OutputInterval | 正整数，指明相隔多少代调用输出函数 | 正整数 \| {1} |
| PlotFcns | 把算法计算出来的数据绘制成图形的函数句柄数组 | @gaplotbestf<br>@gaplotbestgenome<br>@gaplotdistance<br>@gaplotexpectation<br>@gaplotgeneology<br>@gaplotselection<br>@gaplotrange<br>@gaplotscorediversity<br>@gaplotscores<br>@gaplotstopping \| {[ ]} |
| PlotInterval | 正整数，指明调用图形函数间隔的代数 | 正整数 \| {1} |
| PopulationSize | 种群尺度 | 正整数 \| {20} |
| SelectionFcn | 选择进行交叉或变异的父辈的函数句柄 | @selectiongoldberg<br>@selectionrandom<br>{@selectionstochunif}<br>@selectionroulette<br>@selectiontournament |
| StallGenLimit | 正整数，停滞代数限，如果相隔 StallGenLimit 代，目标函数没有改进，则算法停止 | 正整数 \| {50} |
| StallTimeLimit | 正标量，停滞时间限，如果在 StallTimeLimit 秒后，目标函数没有改进，则算法停止 | 正标量 \| {20} |
| TimeLimit | 正标量，时间限，算法运行 TimeLimit 秒后停止 | 正标量 \| {30} |
| Vectorized | 字符串，指明运算的适应度函数是否是向量 | 'on' \| {'off'} |

参见：gaoptimget，gatool。

**4．命令 gatool**

功能：打开遗传算法工具。

格式：

```
gatool
```

详细说明：打开遗传算法工具，给遗传算法提供图形用户界面。

可使用遗传算法工具来运行遗传算法，求解优化问题，并显示结果。

参见：ga，gaoptimset。

### 8.4.3 标准算法选项

遗传算法工具提供了许多标准算法选项，见表8.3。

表**8.3** 遗传算法工具标准算法选项

| 步　骤 | 算法选项 |
|---|---|
| 初始化种群 | uniform |
| 适应度计算 | rank based，proportional，top(truncation)，linear scaling and shift |
| 选择 | roulette，stochastic uniform selection(SUS)，tournament，uniform |
| 交叉 | heuristic，intermediate，scattered，single-point，two-point |
| 变异 | gaussian，uniform multipoint |
| 绘图 | best fitness，best individual，distance among individuals，expectation of individuals，range，diversity of population，selection index，stopping conditions |

遗传算法可以帮助我们确定这种具有多个局部最小值函数的最优解。

遗传算法工具提供了通过使用所有关键部件(包括使用算法选项)来定义问题的能力，实现了运行时间的可视化。

所谓运行时间可视化，是指函数正在优化时所产生的结果可以通过使用遗传算法工具中的绘图函数用图形表示出来，并且运行时间的可视化可以通过使用遗传算法工具中的绘图函数在该函数优化的同时产生。

使用遗传算法工具时常常需要指定：

(1) 群体大小。

(2) 优良的子辈个数。

(3) 交叉片段(Crossover fraction)。

(4) 子群体间迁移(Migration among subpopulations)，使用环形拓扑。

通过提供用户自定义函数可以定制这些算法选项，并且可以用不同的数据格式来描述问题，比如使用混合整型数或复数作为变量。可以以时限、停滞时限、适应度界限、繁殖辈数等为基准来作为算法的终止准则。最后，可以通过矢量化适应度函数来提高运算速度。

# 第九章　使用 MATLAB 直接搜索工具

MathWorks 公司最新发布的 MATLAB 7.0 Release 14 中包含一个专门设计的遗传算法与直接搜索工具箱(Genetic Algorithm and Direct Search Toolbox)，这个工具箱集成有遗传算法工具和直接搜索工具。

上一章已经对遗传算法与直接搜索工具箱进行了概要介绍，对其中的遗传算法工具及其使用方法进行了详细叙述。本章则集中介绍其中的直接搜索工具及其使用方法。

## 9.1　直接搜索工具概述

与遗传算法一样，直接搜索算法工具也可以作为其他优化方法的一个补充，用来寻找最佳起始点，继而可以通过使用传统的优化技术来进一步找出最优解。

直接搜索工具函数可以通过图形界面或 MATLAB 命令行来访问。所有算法工具函数都是用 MATLAB 语言编写的，对用户开放，因此可以查看算法、修改源代码或生成用户函数。

工具箱中的直接搜索算法实现的是模式搜索方法。大多数传统优化方法通过使用梯度或高阶导数的方法来搜寻优化点，而模式搜索方法可搜索出最大最小正基模式。它可以处理边界约束、线性等式、线性不等式，并且不需要函数可微或连续。

模式搜索算法包括下列选项：

(1) 表决方法：用来决定怎样产生和估计模式中的点数以及每一步生成的最大点数。可以通过控制点的表决顺序来提高效率。

(2) 搜索方法：用来选择比表决步效率更高的搜索方法。可以在一个模式内或整个搜索空间中执行搜索。与遗传算法相似，全局搜索方法可以用来获得一个好的起始点。

(3) 网格：用来表示迭代过程中控制模式如何变化。对于各维尺度不同的问题，还需要调整网格。可以选择初始网格尺寸、网格细分因子或网格收缩因子。网格加速器加快了最小值附近的收敛速度。

(4) 缓存器：用来存储优化过程中计算量大的目标函数的估计点。可以指定模式搜索算法中缓存器的大小和容差，还可以在计算过程中调整缓存器容差，以提高寻优的速度和效率。

通过命令行或图形用户界面可修改上述任一选项。

## 9.2 直接搜索算法

本节介绍直接搜索与模式搜索的基本概念，说明工具箱中完成模式搜索的主函数，提供一个用模式搜索来求解优化问题的实例，解释模式搜索的一些基本术语，阐明直接搜索算法的工作原理与工作过程，展示如何绘制目标函数的值和模式搜索产生的点序列的网格尺寸的图形。

### 9.2.1 何谓直接搜索

直接搜索(Direct Search)是一种求解优化问题的方法，它不要求任何目标函数梯度的信息。与使用梯度或高阶导数信息来搜索优化点的较传统的优化算法相反，直接搜索算法搜索当前点周围的一系列点，寻找目标函数值低于当前点值的那些点。我们可以使用直接搜索算法来求解那些目标函数不可微、甚至不连续的问题。

直接搜索工具实现一类特殊的直接搜索算法，称为模式搜索(Pattern Search)算法。模式搜索算法确定一个点的序列，这个点序列呈现越来越接近理想点的趋势。在每一步，该算法搜索在当前点周围的一系列点，称为网格(Mesh)。当前点是指该算法在前一步计算出来的点。该算法通过把当前点与一个称为模式(Pattern)的固定向量集的标量倍数相加来构成网格。如果算法在网格中找到一个新点，且在该点比在当前点使目标函数得到改善，则该算法在下一步就将新点作为当前点。

### 9.2.2 执行模式搜索

本节简要介绍模式搜索工具和实现模式搜索的图形用户界面(GUI)，主要内容包括：从命令行调用模式搜索和使用模式搜索工具 GUI。

**1.** 从命令行调用模式搜索

为了在命令行上对无约束问题执行模式搜索，可用下面的语法调用函数 patternsearch：

```
[x fval]=patternsearch(@objfun, x0)
```

其中：@objfun——目标函数句柄；x0——模式搜索的起始点。返回的结果为：fval——目标函数的最终值；x ——获得最终值的点。

9.3.2 节“从命令行运行模式搜索”将详细解释如何使用函数 patternsearch。

**2.** 使用模式搜索工具 **GUI**

键入 psearchtool，打开模式搜索工具 GUI，如图 9.1 所示。

为了使用模式搜索工具，首先必须输入下列信息：

(1) Objective function(目标函数)——想要最小化的目标函数。输入目标函数的形式为@objfun，这里 objfun.m 是计算目标函数的一个 M 文件，符号“@”产生 objfun 的函数句柄。

(2) Start point(起始点)——即开始点。算法从该点开始进行优化。

在“Constraints(约束)”窗格中输入对问题的约束。如果问题是无约束的，则该字段应保持为空。

然后，单击“Start(开始)”按钮，模式搜索工具随即在“Status and results(状态与结果)”窗格中显示出优化的结果。

我们也可以在“Options(选项)”窗格中改变模式搜索的选项。为了查看某一类选项，可单击与之相关的加号“+”。

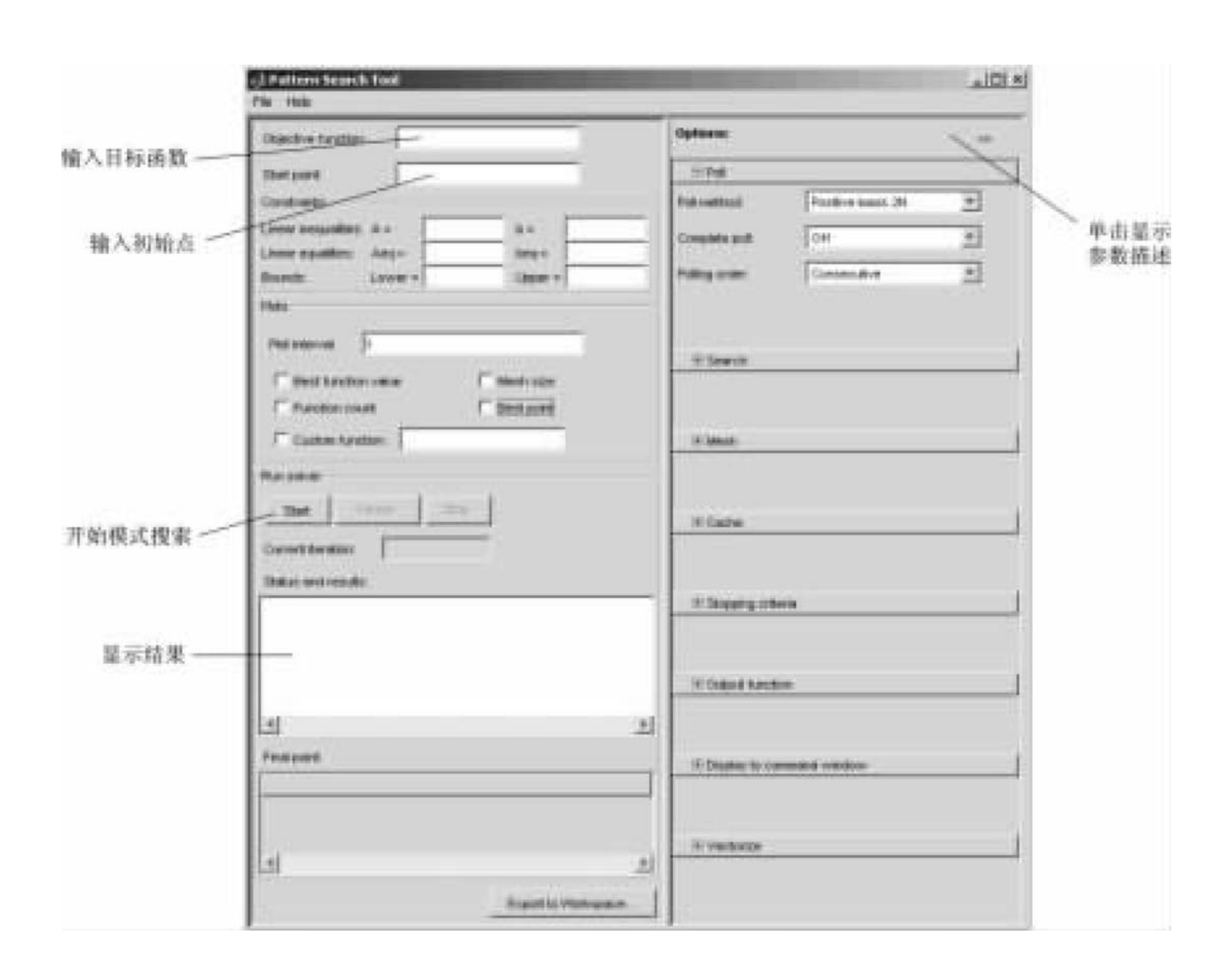



图 9.1　模式搜索工具

## 9.2.3　寻找函数最小值

本节介绍使用模式搜索工具来寻找函数最小值的一个实例。

**1.** 目标函数

本例使用目标函数 ps_example，它包含在遗传算法与直接搜索工具箱之中。键入下列命令就可以查看该函数的代码：

```
type ps_example
```

图 9.2 显示出了该函数的图形细节。

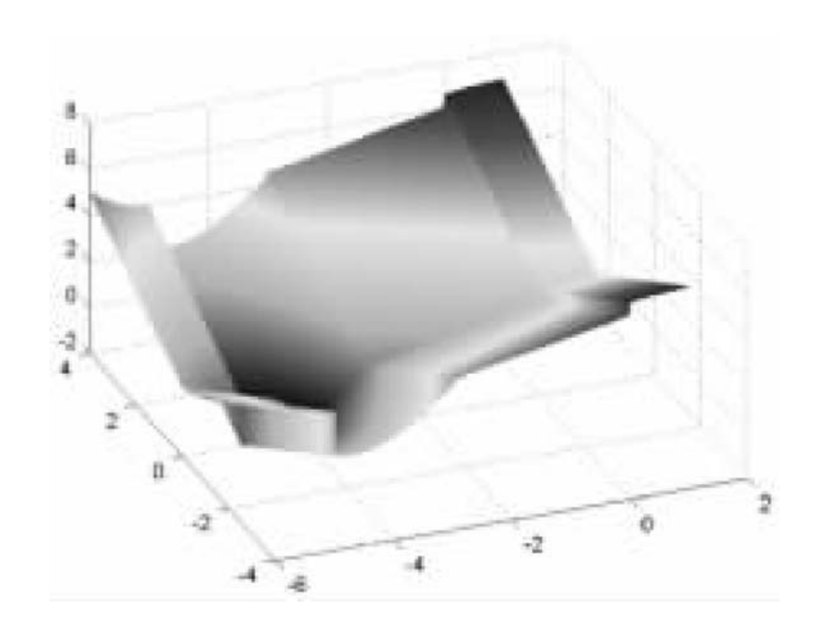


图 9.2　目标函数 ps_example

**2．寻找函数的最小值**

为了寻找 ps_example 的最小值，进行如下步骤：

(1) 键入 psearchtool，打开模式搜索工具。

(2) 在模式搜索工具的“Objective function(目标函数)”文本框中输入@ps_example。

(3) 在“Start point(起始点)”文本框中输入[2.1 1.7]。

打开模式搜索工具后的这两步操作，即输入目标函数与起始点，如图 9.3 所示。因为该问题是无约束的，所以 Constraints 字段保持为空。




图 9.3　输入目标函数与起始点

(4) 单击“Start”按钮，运行模式搜索。在“Status and results”窗格中显示模式搜索的结果，如图 9.4 所示。

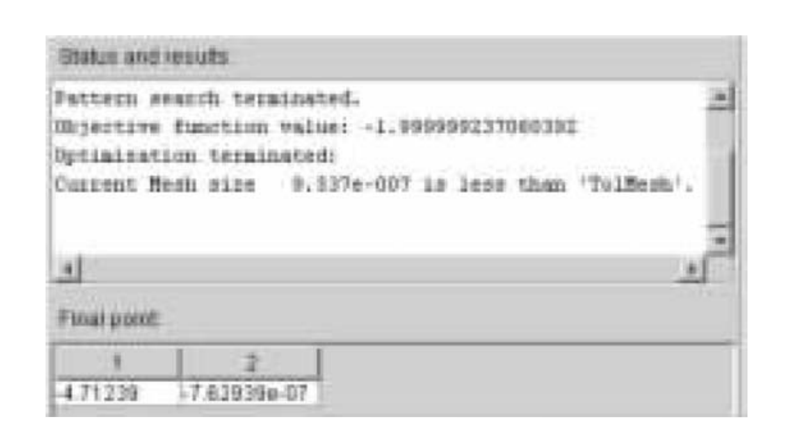



图 9.4　目标函数 ps_example 的模式搜索结果

函数的最小值约为－2。“Final point(最终点)”窗格中显示最小值出现的那个点。

**3.** 绘制目标函数值和网格尺寸的图形

为了查看模式搜索的性能，可以显示每一次迭代时的最佳函数值与网格尺寸。首先，在“Plots(绘图)”窗格中选中 Best function value(最佳函数值)和 Mesh size(网格尺寸)复选框，如图 9.5 所示。

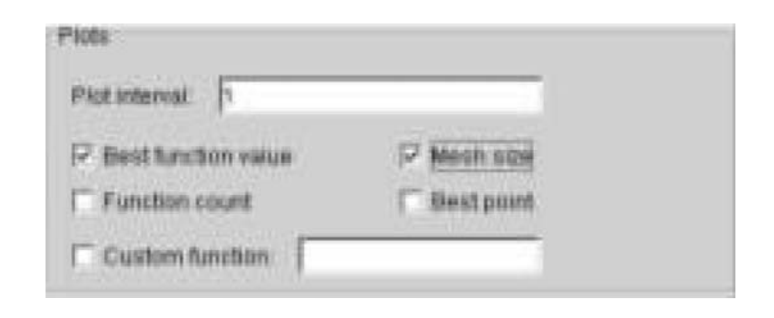



图 9.5 绘图窗口

然后，单击“Start”按钮，运行模式搜索。这时，会显示出图 9.6 所示的细节。

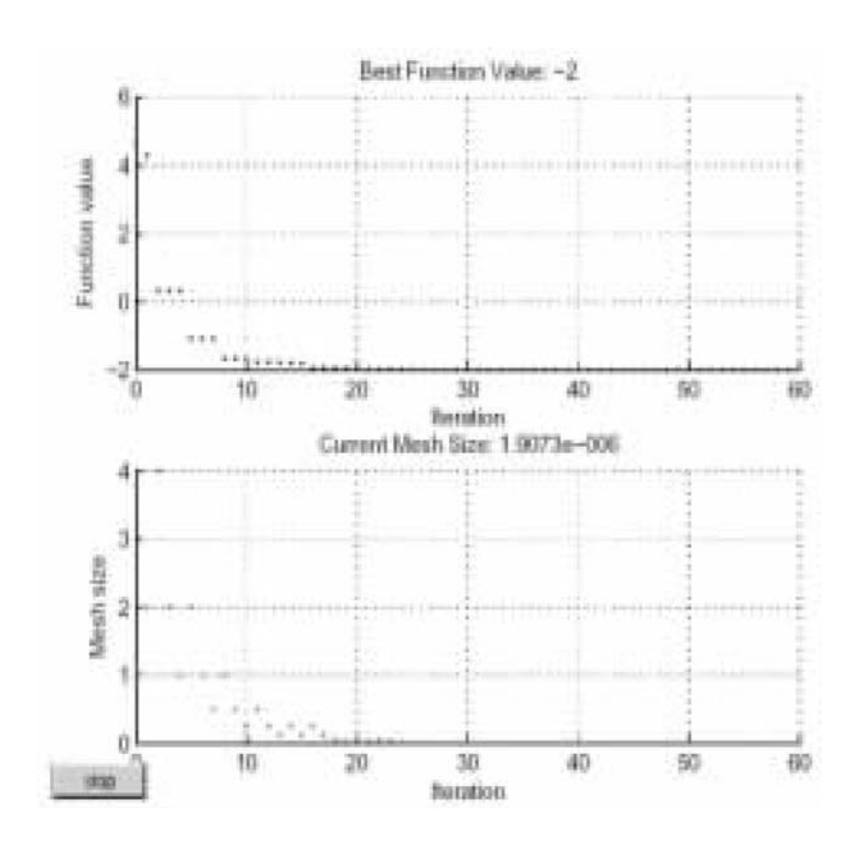



图 9.6 ps_example 的最佳函数值与网格尺寸

图 9.6 上半部分显示出在每一次迭代时最佳点的目标函数值。典型情况下，目标函数值在早期迭代时迅速改善，然后，随着它们逼近理想值而变平、稳定。

图 9.6 下半部分显示出在每一次迭代时的网格尺寸。网格尺寸在每一次成功的迭代后增加，在每一次不成功的迭代后减少。9.2.5 节“模式搜索如何工作”中对此进行了解释。

## 9.2.4 模式搜索术语

本节解释模式搜索的 3 个标准术语，包括模式(Pattern)、网格(Mesh)和表决(Poll)。

**1.** 模式

模式是若干向量的一个集合，向量由算法来使用，以确定在每次迭代时要搜索哪一点。例如，如果在某优化问题中存在两个独立变量，则缺省模式由下面的向量组成：

$v_1=[1\ 0]$

$v_2=[0\ 1]$

$v_3=[-1\ 0]$

$v_4=[0\ -1]$

图 9.7 显示出了这些向量。

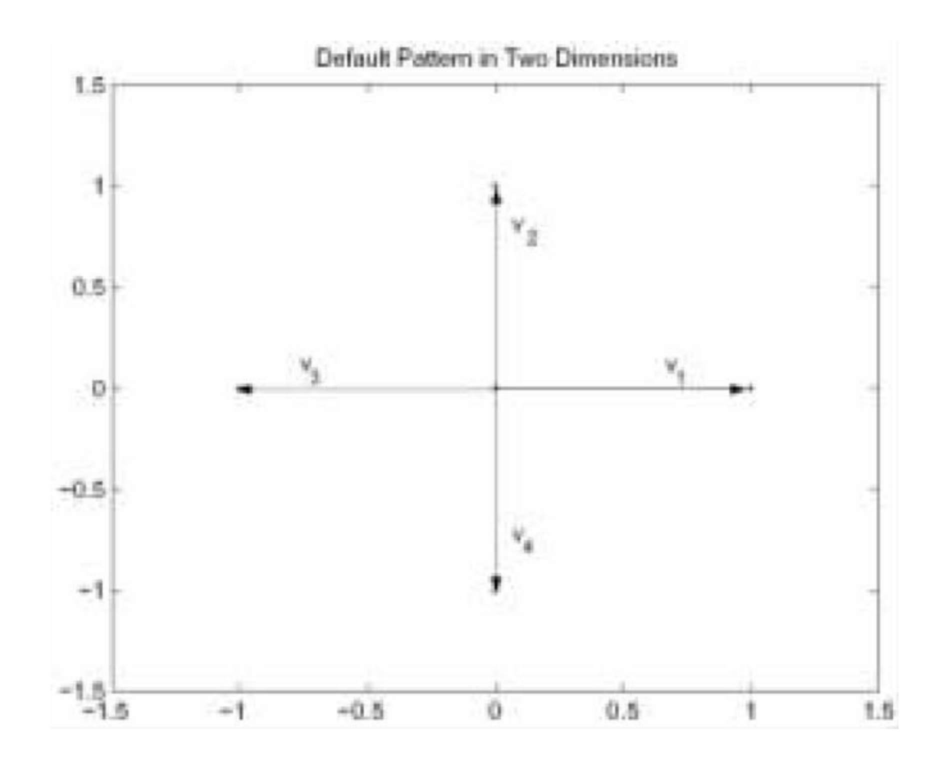



图 9.7　二维的缺省模式向量

**2．** 网格

在每一步中，模式搜索算法搜索一组点(称为网格(mesh))，以寻找使目标函数得到改善的点。该算法形成网格的方法是：

(1) 给模式向量乘以一个标量(称为网格尺寸(Mesh Size))。

(2) 把结果向量与当前点相加。当前点是指在前一步找到的具有最佳目标函数值的点。

例如，假定当前点是[1.6 3.4]，模式由下列向量组成：

$v_1=[1\ 0]$

$v_2=[0\ 1]$

$v_3=[-1\ 0]$

$v_4=[0\ -1]$

当前网格尺寸是 4。算法给模式向量乘以 4，并且把它们与当前点相加，从而获得下面的网格：

$[1.6\ 3.4]+4*[1\ 0]=[5.6\ 3.4]$

$[1.6\ 3.4]+4*[0\ 1]=[1.6\ 7.4]$

$[1.6\ 3.4]+4*[-1\ 0]=[-2.4\ 3.4]$

$[1.6\ 3.4]+4*[0\ -1]=[1.6\ -0.6]$

产生一个网格点的模式向量也称为该网格点的方向(Direction)。

**3. 表决**

在每一步中，算法通过计算它们的目标函数值，在当前网格中表决选出若干点。当选项“Complete poll(完全表决)”具有缺省设置 Off 时，算法一旦找到目标函数值小于当前点函数值的那个点就停止对网格点的表决选取。如果这种情形出现，则为表决成功，并且所找到的点在下一次迭代时变成当前点。

注意：算法只计算当前网格中的点的目标函数值，直到找到满足停止表决条件的某点为止。

如果算法未能找到使目标函数得到改进的点，则表决不成功，并且在下一次迭代时当前点保持不变。

如果选项“Complete poll”设置为 On，算法计算所有网格点的目标函数值，然后，把具有最小目标函数值的网格点与当前点进行比较。如果该网格点比当前点具有更小的目标函数值，则表决成功。

## 9.2.5 模式搜索如何工作

模式搜索算法可以找到一个点序列 x0，x1，x2，…，它们逐渐逼近理想点，序列中的每一点到下一点的目标函数值逐渐减少。本节以前面所描述的目标函数 ps_exampl 为例，说明模式搜索算法是如何工作的。

为了使解释简明起见，假定在这里描述模式搜索算法如何工作的时候，选项“Mesh”中的参数“Scale”设置为 Off。

**1. 成功选举**

对于目标函数 ps_example，模式搜索从所提供的起始点 x0 开始。在本例中，起始点 x0＝[2.1 1.7]。下面叙述模式搜索算法的工作过程。

1）第 1 次迭代

在第 1 次迭代时，网格尺寸为 1，模式搜索算法把模式向量与起始点 x0＝[2.1 1.7]相加，计算出下列网格点：

[1 0]＋x0＝[3.1 1.7]

[0 1]＋x0＝[2.1 2.7]

[－1 0]＋x0＝[1.1 1.7]

[0 －1 ]＋x0＝[2.1 0.7]

算法以上面显示的顺序计算各网格点的目标函数。图 9.8 显示出 ps_example 在起始点和各网格点的值。

该算法通过计算各网格点的目标函数值来表决选取各网格点，直到它找到其函数值小于 x0 点的函数值 4.6347 的点。在本例中，它找到的第一个这样的点是[1.1 1.7]，目标函数在该点的值是 4.5146，故迭代 1 的选取获得成功。算法设置序列中的下一点为

x1＝[1.1 1.7]

注意：按照缺省值，模式搜索算法一旦找到其目标函数值小于当前点的一个网格点时，就停止当前的迭代。因此，该算法不可能选出所有的网格点。我们可以通过设置“Complete poll”选项为 On，而使算法选取所有网格点。

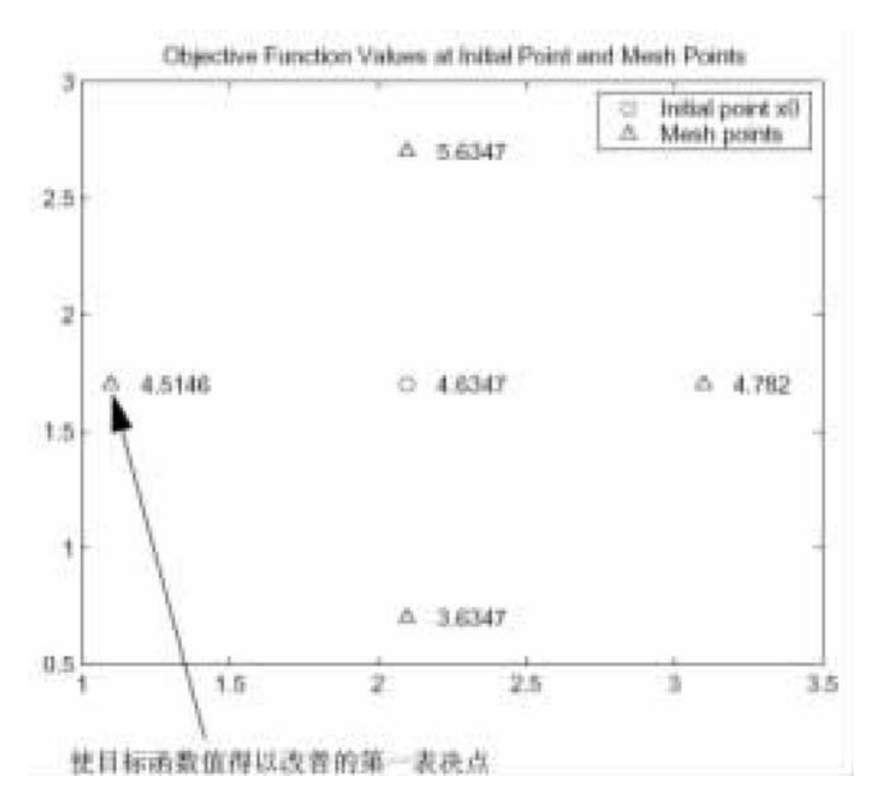



图 9.8　目标函数在起始点和各网格点的值

2）第 2 次迭代

在一次成功选举之后，该算法把当前网格尺寸乘以 2，这是“Mesh”选项窗格中“Expansionfactor(扩张因子)”的缺省值。因为初始网格尺寸是 1，在第 2 次迭代时网格尺寸为 2，所以第 2 次迭代时的网格包含下列点：

2 * [1 0]+x1=[3.1 1.7]

2 * [0 1]+x1=[1.1 3.7]

2 * [−1 0]+x1=[−0.9 1.7]

2 * [0 −1 ]+x1=[1.1 −0.3]

目标函数 ps_example 在当前点 x1 和各网格点的相应值如图 9.9 所示。

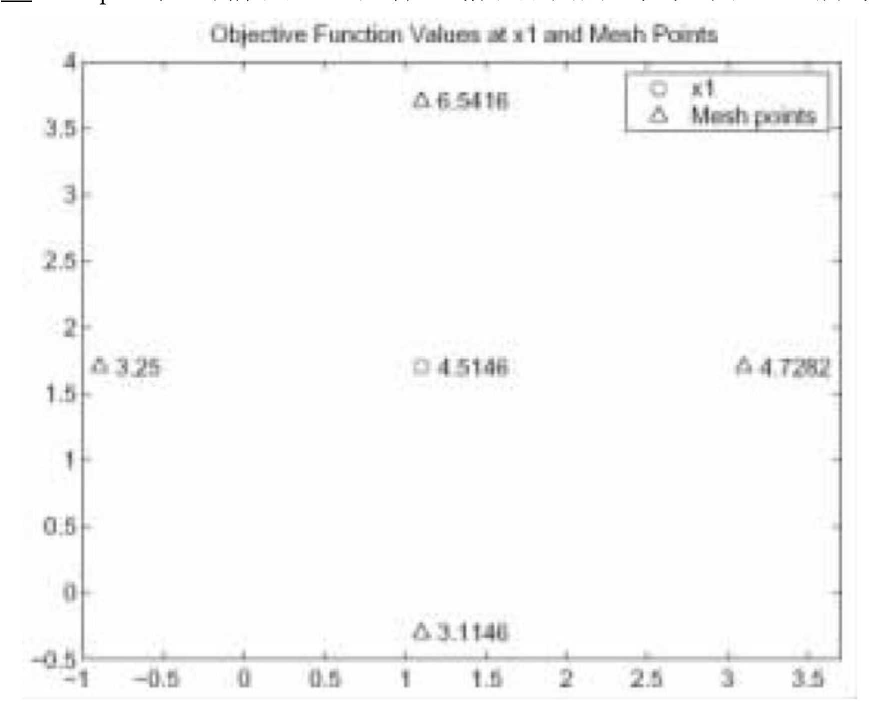



图 9.9　目标函数在 x1 点和各网格点的值

算法检测各网格点，寻找其函数值小于当前点 x1 处的值 4.5146 的一个点。它找到的第一个满足条件的点是[－0.9 1.7]，目标函数在该点处的值是 3.25，故迭代 2 的选取再次成功。该算法设置序列中的第二个点为

x2＝[－0.9 1.7]

因为选取成功，算法把当前网格尺寸乘以 2，得到第 3 次迭代的网格尺寸为 4。

3）第 3 次迭代

到第 3 次迭代时，当前点是 x2＝[－0.9 1.7]，且网格尺寸为 4，所以网格由下列点组成：

4＊[1 0]＋x3＝[3.1 1.7]

4＊[－1 0]＋x3＝[－4.9 1.7]

4＊[0 1]＋x3＝[－0.9 5.7]

4＊[0 －1]＋x3＝[－0.9 －2.3]

算法选取各网格点，找到其值小于 x2 处值的第一个点[－4.9 1.7]，该点处目标函数的值是－0.2649，故迭代 3 的选取再次成功。该算法设置序列中的第三个点为

x3＝[－4.9 1.7]

因为选取成功，算法把当前网格尺寸再乘以 2，得到第 3 次迭代后的网格尺寸为 8。

**2．**不成功选举

到第 4 次迭代时，当前点是 x3＝[－4.9 1.7]，网格尺寸为 8，所以网格由下列点组成：

8＊[1 0]＋x3＝[3.1 1.7]

8＊[－1 0]＋x3＝[－12.9 1.7]

8＊[0 1]＋x3＝[－4.9 9.7]

8＊[0 －1]＋x3＝[－4.9 －6.3]

图 9.10 显示出了各网格点以及它们的目标函数值。

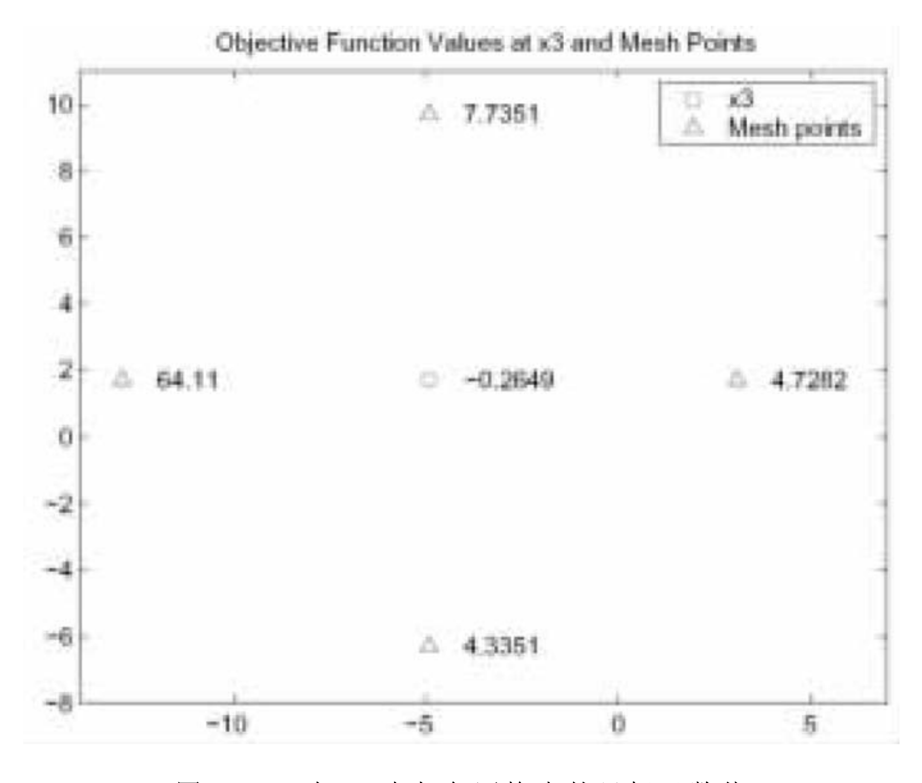



图 9.10　在 x3 点与各网格点的目标函数值

这次迭代，不存在其目标函数值小于 x3 处值的网格点，故选取不成功。在本例中，算法不改变下次迭代的当前点，即 x4＝x3。

下次迭代时，算法对当前网格尺寸乘以 0.5，这是“Mesh”窗格中的“Contraction factor（收缩因子）”选项的缺省值。因此，下次迭代的网格尺寸为 4。这样，算法的选举具有了较小的网格尺寸。

**3. 显示每次迭代的结果**

通过设置在“Display to command window（显示到命令窗口）”选项中的“Level of display（显示级别）”为“Iterative（迭代）”，就可以显示出每次迭代时模式搜索的结果。这使得我们可以评估模式搜索的进展，在必要时可以改变相关选项。

有了这样的设置，模式搜索将在命令行显示每次迭代的信息。对于本例，显示的前几行如下：

| Iter | f-count | MeshSize | f(x) | Method |
|---|---|---|---|---|
| 0 | 1 | 1 | 4.635 | Start Iterations |
| 1 | 4 | 2 | 4.515 | Successful Poll |
| 2 | 7 | 4 | 3.25 | Successful Poll |
| 3 | 10 | 8 | －0.2649 | Successful Poll |
| 4 | 14 | 4 | －0.2649 | Refine Mesh |

在 Method（方法）之下显示的信息“Successful Poll”表明当前迭代是成功的。例如，第 2 次迭代的选举是成功的。因此，在第 2 次迭代中计算出来的那一点的显示在 f(x)下面的目标函数值小于在第 1 次迭代中计算出来的值。

在第 4 次迭代中，“Method”之下显示的信息“Refine Mesh（改善网格）”告诉我们，选举是不成功的。因此，第 4 次迭代的函数值仍然保持第 3 次迭代的函数值不变。

注意：模式搜索在每次成功选举之后使网格尺寸加倍，而在每次不成功的选举之后使网格尺寸减半。

**4. 更多的迭代**

对于本例，模式搜索在停止之前执行了 88 次迭代。图 9.11 显示出模式搜索的前 13 次迭代计算序列中的若干点。各个点下面的数字表示该算法找到该点时的迭代次数。这个图只显示出与成功选举相对应的迭代次数，因为在一次不成功的选举之后最佳点不发生变化。例如，第 4 次迭代和第 5 次迭代的最佳点就与第 3 次迭代的最佳点相同。

**5. 模式搜索的停止条件**

模式搜索算法运行的停止要遵循一定的停止标准。这些标准列写在模式搜索工具的“Stopping criteria（停止标准）”窗格中，如图 9.12 所示。

当下列任何一个条件满足时，模式搜索算法停止：

（1）网格尺寸小于“Mesh tolerance（网格容限）”。

（2）算法执行的迭代次数达到“Max iteration（最大迭代）”规定的值。

（3）算法完成的目标函数评估的总次数达到“Max function evaluations（最大函数评估）”规定的值。

（4）一次成功选举所找到的点与下一次成功选举所找到的点之间的距离小于“X tolerance（X 的容限）”。

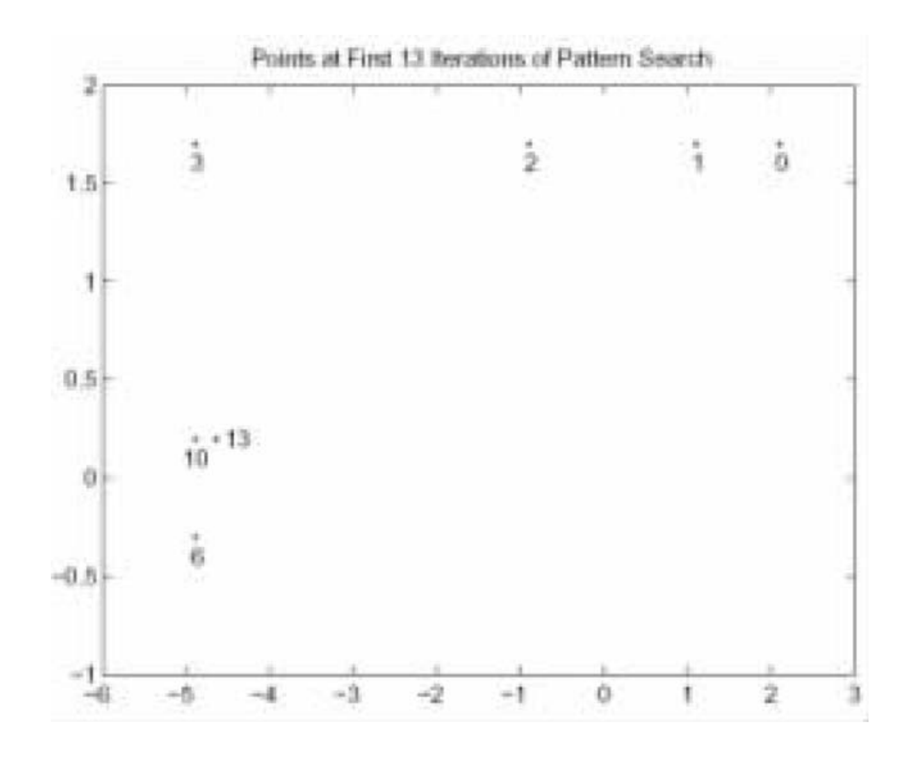



图 9.11　模式搜索在前 13 次迭代的点

Stopping criteria

| | |
|---|---|
| Mesh tolerance: | 1e-006 |
| Max iteration: | 100*numberofvariables |
| Max function evaluations: | 2000*numberofvariables |
| Bind tolerance: | 0.001 |
| X tolerance: | 1e-006 |
| Function tolerance: | 1e-006 |

图 9.12　停止标准窗口

(5) 从一次成功选举到下一次成功选举的目标函数的改变小于“Function tolerance(函数容限)”。

“Bind tolerance(绑定容限)”选项，用来识别有约束问题的现行约束，并不用来作为一种停止标准。

# 9.3　使用直接搜索工具

本节详细介绍直接搜索工具及其使用方法，主要包括：浏览模式搜索工具、从命令行运行直接搜索、模式搜索算法举例和参数化函数。

## 9.3.1　浏览模式搜索工具

**1. 打开模式搜索工具**

在 MATLAB 工作区输入 psearchtool，将打开模式搜索工具，初始界面如图 9.13 所示。模式搜索工具的右半部分是选项参数区，包含各种选项窗格，单击与之相关的加号

“＋”，可以将其展开，方便地进行参数设置。其左半部分是操作显示区，包括各种运行参数的设置及各种状态、结果的显示，从上到下依次为：Objective function（目标函数）、Start point（起始点）、Constraints、Plots、Run solver（运行求解器）、Status and results、Final point，最下面还有一个命令按钮“Export to Workspace”。



图 9.13　模式搜索工具初启时的界面

**2．在模式工具中定义问题**

用模式搜索工具求解问题时，要在下列区域中定义要解决的问题：

（1）Objective function——欲最小化的目标函数。在此输入要计算目标函数的 M 文件的函数句柄。

（2）Start point——模式搜索算法的起始点。

在模式搜索工具中定义待求解的问题，亦即设置目标函数和起始点，如图 9.14 所示。

图 9.14　目标函数、起始点和约束条件输入窗口

注意：在运行模式搜索工具时，不要使用“Editor/Debugger(编辑/调试)”功能来调试目标函数的M文件。如果这样做，在命令窗口会产生Java的多余信息，并使得调试更加困难。因而应该从命令行直接调用目标函数，或者把它传递给patternsearch函数。为了快速调试，可以从模式搜索工具把问题输出到MATLAB工作空间。

对于有约束条件的问题，要在“Constraints”窗格中设置好相应的约束条件。在“Constraints”窗格中可以输入任何约束条件：

(1) Linear inequalities(线性不等式)——在下列文本框中输入不等式约束条件($A\cdot x\leqslant b$)：

- 在文本框“A＝”中输入矩阵A。
- 在文本框“b＝”中输入矩阵向量b。

(2) Linear equalities(线性等式)——在下列文本框中输入等式约束条件($Aeq\cdot x=beq$)：

- 在文本框“Aeq＝”中输入矩阵Aeq。
- 在文本框“beq＝”中输入矩阵向量beq。

(3) Bounds(边界)——在下列文本框中输入边界约束条件($lb\leqslant x$，$x\leqslant ub$)：

- 在文本框“Lower＝”中输入向量的下界lb。
- 在文本框“Upper＝”中输入向量的上界ub。

没有对应约束条件的文本框设置为空。

**3.** 运行模式搜索

在“Run solver(运行求解器)”窗格中单击“Start”按钮，运行模式搜索。这时，“Current iteration(当前迭代)”区域显示当前迭代次数，“Status and results”窗格中显示信息“Pattern search running(模式搜索运行)”。

当模式搜索停止时，“Status and results”窗格中显示：

(1) 信息“Pattern search running”。

(2) 最终点的目标函数值。

(3) 模式搜索停止的原因。

(4) 最终点的坐标。

运行模式搜索算法来求解前面叙述的目标函数ps_example时，所显示出来的信息如图9.15所示。

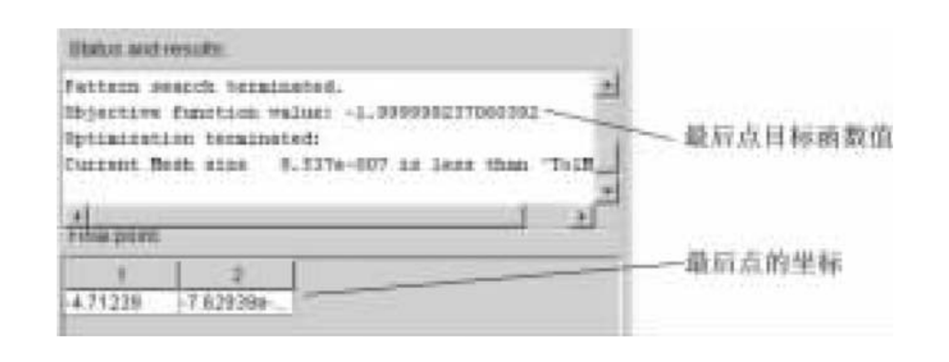



图9.15　模式搜索算法显示的状态与结果

**4.** 求解有约束条件的问题

这里提供一个模式搜索的例子，是带有约束条件的求函数最小值问题。目标函数为

$$F(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{x} + \boldsymbol{f}^{\mathrm{T}}\boldsymbol{x}$$

式中

$$\boldsymbol{H} = \begin{bmatrix} 36 & 17 & 19 & 12 & 8 & 15 \\ 17 & 33 & 18 & 11 & 7 & 14 \\ 19 & 18 & 43 & 13 & 8 & 16 \\ 12 & 11 & 13 & 18 & 6 & 11 \\ 8 & 7 & 8 & 6 & 9 & 8 \\ 15 & 14 & 16 & 11 & 8 & 29 \end{bmatrix}$$

$$\boldsymbol{f} = [20 \quad 15 \quad 21 \quad 18 \quad 29 \quad 24]$$

约束条件为

$$\begin{cases} \boldsymbol{A} \cdot \boldsymbol{x} \leqslant \boldsymbol{b} \\ \boldsymbol{Aeq} \cdot \boldsymbol{x} = \boldsymbol{beq} \end{cases}$$

其中

$$\boldsymbol{A} = [8 \quad 7 \quad 3 \quad 4 \quad 9 \quad 0]$$

$$\boldsymbol{b} = [7]$$

$$\boldsymbol{Aeq} = \begin{bmatrix} 7 & 1 & 8 & 3 & 3 & 3 \\ 5 & 0 & 5 & 1 & 5 & 8 \\ 2 & 6 & 7 & 1 & 1 & 8 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\boldsymbol{beq} = [84 \quad 62 \quad 65 \quad 1]$$

运行模式搜索工具来求解这个问题，先输入 psearchtool，打开模式搜索工具。然后设置“Objective function”为@lincontest7——工具箱中包含一个计算这个目标函数值的 M 文件 lincontest7.m。因为用矩阵和向量定义起始点和约束条件太大，所以在 MATLAB 工作空间中分别把它们设置为变量，然后在模式搜索工具输入这些变量的名称，这样做是非常方便的。下面输入：

```
x0=[2 1 0 9 1 0];
Aineq=[-8 7 3 -4 9 0];
bineq=[7];
Aeq=[7 1 8 3 3 3; 5 0 5 1 5 8; 2 6 7 1 1 8; 1 0 0 0 0 0];
beq=[84 62 65 1];
```

然后，在模式搜索工具输入下列参数：

（1）设置“Start point”为 x0。

（2）在“Linear inequalities”选项中设置“A=”为 Aineq，设置“b=”为 bineq。

（3）在“Linear equalities”选项中设置“Aeq=”为 Aeq，设置“beq=”为 beq。

在模式搜索工具中设置的这些参数如图 9.16 所示。

最后，单击“Start”按钮，运行模式搜索。算法结束后，在“Status and results”窗口中显

示算法运行的状态与结果，如图 9.17 所示。

图 9.16　设置有约束问题的目标函数、起始点和约束条件

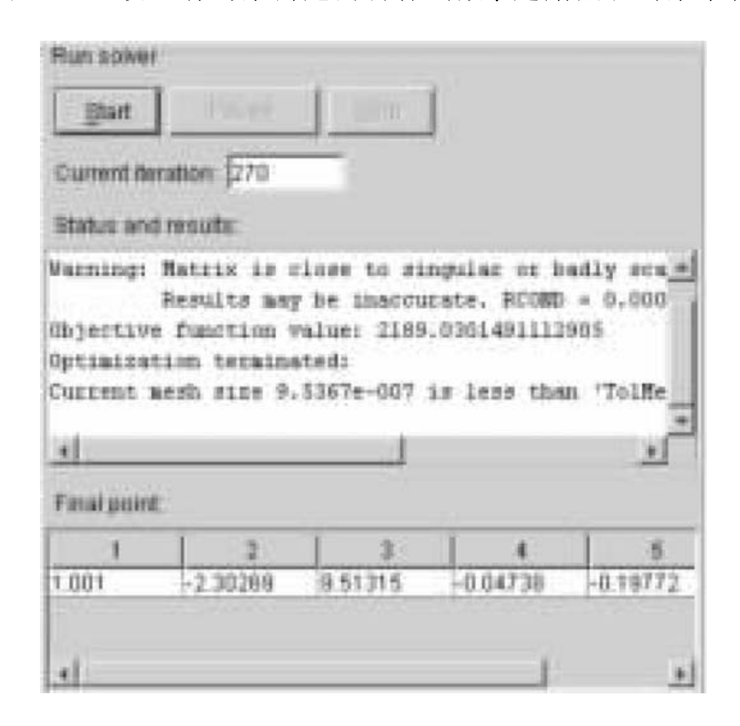



图 9.17　求解有约束问题的状态与结果

**5．**暂停和停止算法

当模式搜索正在运行时，可以按下面的方法暂停或停止算法运行：

(1) 单击“Pause(暂停)”按钮暂停运行，这时按钮的名称更改为“Resume(恢复)”，再单击此按钮，即从暂停处继续运行。

(2) 单击 “Stop(停止)” 按钮停止运行。“Status and results”窗口中显示出单击“Stop”按钮后当前点的目标函数值。

**6．**图形显示

在“Plots(绘图)”窗格(参见图 9.18)中，能够通过选择来控制显示模式搜索运行结果的图形。

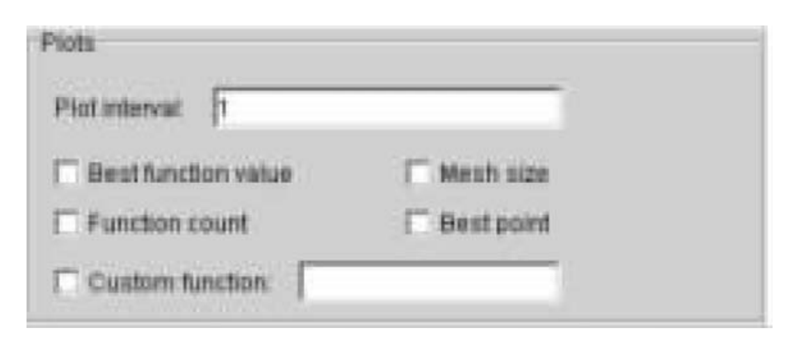



图 9.18　“Plots(绘图)”窗格

选择需要显示图形的复选框，例如，可以选择“Best function value(最佳函数值)”和“Mesh size(网格尺寸)”。运行前面叙述的目标函数 ps_example，所显示的图形如图 9.19 所示。

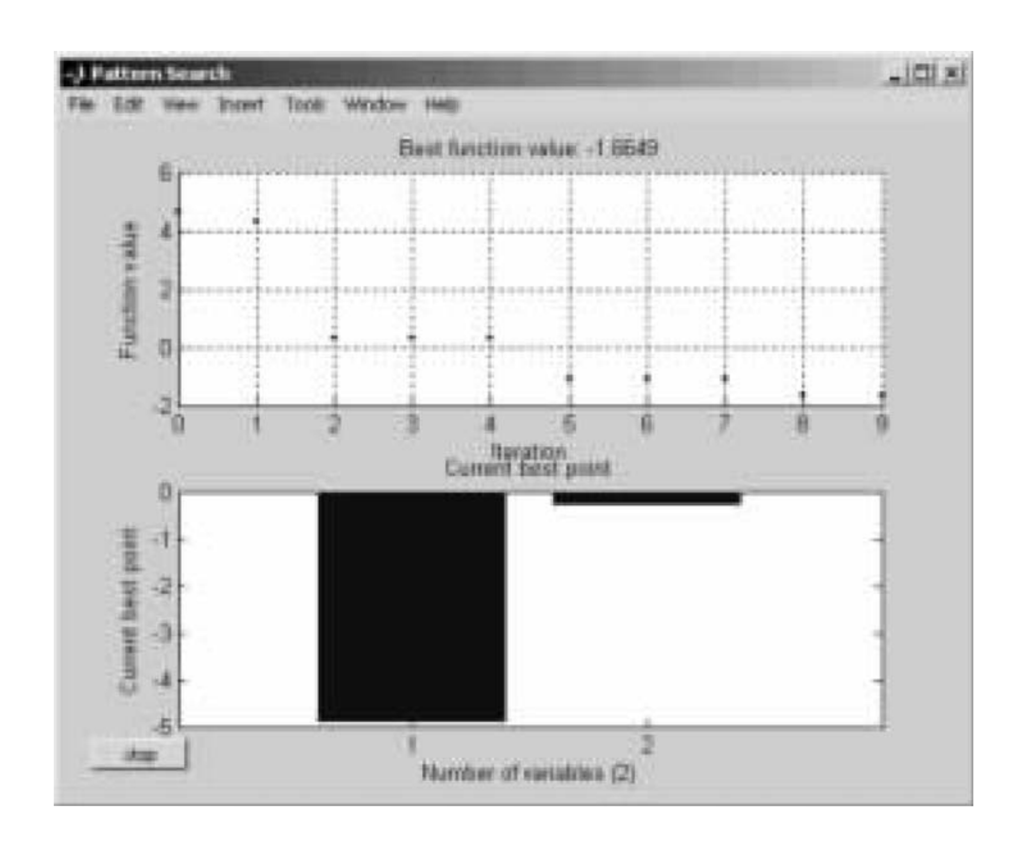



图 9.19　目标函数 ps_example 模式搜索结果

在图 9.19 中，上面显示每一次迭代的目标函数值，下面显示当前迭代的最佳目标函数值对应点的坐标。

注意：需要显示多项参数的图形时，选择该参数项的复选框，就可以在单独窗口的更大区域显示相应的图形。

**7.** 在模式搜索工具中设置参数

设置模式搜索工具使用时的选项参数有两种方法：一种是在模式搜索工具 GUI 的“Options(选项)”窗格中直接进行设置，另一种是在 MATLAB 工作空间中通过命令行方式进行设置。

在“Options”窗格中设置模式搜索的各种运行参数，如图 9.20 所示。每一类参数对应有一个窗格，单击该类参数时，对应窗格展开。例如，点击“Poll”参数选项，表决窗格展开，可以逐一设置其中的参数项，如 Poll method、Complete poll、Polling order 等。

此外，其他选项参数类还有：Search(搜索)、Mesh、Cache(缓存)、Stopping criteria(停止标准)、Output(输出)、Display to command window(显示到命令窗口)、Vectorize(向量化)等。这些选项各自对应一个参数窗格，单击后相应窗格随即展开，可以进行参数项的设置。

所有变量参数的含义及详细描述，可参见 9.4.1 节“模式搜索参数”。

我们可以在 MATLAB 工作空间把参数设置为变量。可以通过在相应的编辑框复制参数的值，或者通过在包含参数值的 MATLAB 工作空间输入变量的名称，直接设置多个变量。在 MATLAB 工作空间定义大矩阵或向量值作为变量，是非常方便的。

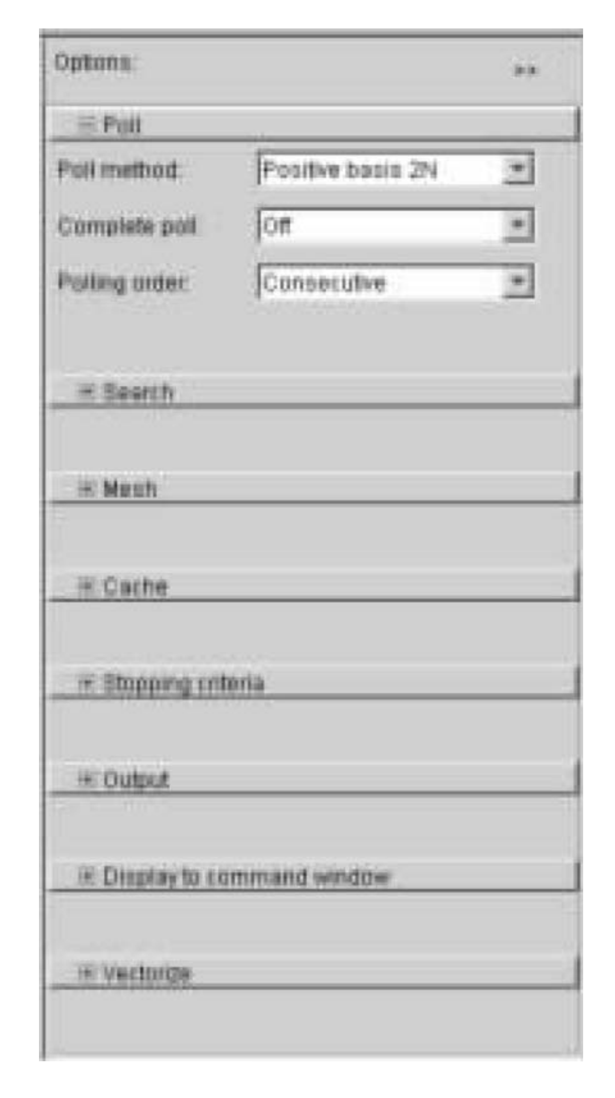



图 9.20　模式搜索选项参数

**8.** 输入、输出参数和问题

参数和问题结构可以从模式搜索工具输出到 MATLAB 工作空间，并且此后在工具箱的工作期间可以把它们导入进来。这样，就为模式搜索工具在以后活动期间保存自己所做的工作提供了一个简单途径。下面描述如何输入、输出参数和问题结构。

1）输出参数、问题和结果

使用模式搜索工具求解一个问题后，可以输出下列信息到 MATLAB 工作空间：

（1）问题定义，包括目标函数、起始点和问题的约束条件。

（2）当前的选项参数。

（3）算法运行的结果。

然后，单击“Export to Workspace(输出到工作空间)”按钮，或者从“File(文件)”菜单中选择“Export to Workspace”菜单项，打开“Export To Workspace”对话框，如图 9.21 所示。

对话框提供下列选项：

（1）在 MATLAB 结构中保存目标函数和参数。选择“Export problem and options to a MATLAB structure named(输出问题与参数到已命名的 MATLAB 结构)”，可以在 MATLAB 结构中保存目标函数和参数，这时，需要在右面的文本框中输入一个 MATLAB 结构的名字。如果需要在当前任务活动期间运行模式搜索，可选择“Include information

图 9.21 输出到工作空间对话框

needed to resume this run(包括所需要的信息以重新开始本次运行)”，这样，最近一次搜索的最终点将保存在“Start point”的位置。如果随后想以最近一次搜索的最终点来运行模式搜索，可使用这个参数。

(2) 在 MATLAB 结构中保存参数。选择“Export options to a MATLAB structure named”，可以在 MATLAB 结构中保存参数。这时，在这个选项后面的文本框中，输入该参数结构的名字。

(3) 在 MATLAB 结构中保存算法的最后运行结果。选择“Export results to a MATLAB structure named”，可在 MATLAB 结构中保存算法的最后运行结果。这时，在这个选项后面的文本框中，输入这个结果结构的名称。

2) 运行模式搜索来输出问题

输出问题的描述可参见前面求解有约束条件的问题的例子。对于这个例子，可在命令行调用 patternsearch 函数，运行模式搜索。其具体步骤为：

(1) 打开“Export To Workspace”对话框。

(2) 在“Export To Workspace” 对话框的“Export problem and options to a MATLAB structure named”右面的文本框中，输入问题结构的名字，例如 my_psproblem。

(3) 用 my_psproblem 调用 patternsearch 函数：

```
[x fval]=patternsearch(my_psproblem)
```

结果为

```
x=
    1.0010   -2.3027   9.5131   -0.0474   -0.1977   1.3083
fval=
    2.1890e+003
```

3) 输入参数

为了从 MATLAB 工作空间为模式搜索输入参数结构，可以从“File”菜单中选择“Import Options(输入参数)”，打开一个对话框，在 MATLAB 工作空间，显示一系列有效的模式搜索的参数结构。当选择参数结构，并单击“Import”按钮时，模式搜索工具重新设置它的参数为所输入结构的值。

注意：不能在包含任何无效参数区域输入参数结构。无效区域结构在“Import Pattern Search Options(输入模式搜索参数)”对话框中不显示。

创建参数结构有两种方法：

(1) 用 options 调用 psoptimset，作为输出。

(2) 在模式搜索工具中，从“Export To Workspace”对话框保存当前参数。

4) 输入问题

为了输入以前从模式搜索工具输出的问题，可以从“File”菜单中选择“Import Problem”菜单项，打开一个对话框，在 MATLAB 工作空间，显示一系列有效的模式搜索问题结构。当选择了一个问题结构并单击“OK”按钮时，模式搜索工具在输入结构内重新设置问题的定义和参数值。另外，当创建问题结构时，选择“Include information needed to resume this run(包括所需要的信息以重新开始本次运行)”，则工具箱把最后一次运行的最终点设置为下一次运行的“Start point”，它比输出结构优先。

**9. 生成一个 M 文件**

为了生成一个运行模式搜索的 M 文件，可以从“File”菜单中选择“Generate M-File(生成 M 文件)”菜单项来进行，生成的 M 文件可保存在 MATLAB 路径的目录中。这个 M 文件使用在模式搜索工具中指定的目标函数和参数。在命令行调用这个 M 文件时，所返回的结果与生成 M 文件时模式搜索工具在同一位置使用目标函数和参数设置所返回的结果相同。

## 9.3.2 从命令行运行模式搜索

使用模式搜索工具的另一个途径是在命令行调用 patternsearch 函数。本节详细叙述这种使用方法，内容包括：使用缺省参数值调用 patternsearch 函数；在命令行设置 patternsearch 的参数；从模式搜索工具使用参数和问题。

**1. 使用缺省参数值调用 patternsearch 函数**

这里描述如何使用缺省参数来运行模式搜索。

1) 无约束条件问题的模式搜索

对于无约束条件问题，用下面语句调用 patternsearch 函数：

```
[x fval]=patternsearch(@objectfun, x0)
```

其中，输出变量为：

- x——最终点。
- fval——目标函数在 x 点的值。

需要输入的变量为：

- @objectfun——目标函数 objectfun 的句柄，可以写成一个 M 文件。参见 8.1.2 节“编写待优化函数的 M 文件”。
- x0——模式搜索算法的起始点。

对于前面描述的例子，即目标函数为 ps_example，在这里，要从命令行运行模式搜索，可输入

```
[x fval]=patternsearch(@ps_example, [2.1 1.7])
```

运行结果为

```
Optimization terminated:
Current mesh size 9.5367e-007 is less than 'TolMesh'.
x=
```

```
    -4.7124    -0.0000
fval=
    -2.0000
```

2）有约束条件问题的模式搜索

如果问题带有约束条件，则应使用语句

```
[x fval]=patternsearch(@objfun, x0, A, b, Aeq, beq, lb, ub)
```

其中：

- A 和 b 分别为代表不等式约束条件的矩阵和向量($A \cdot x \leqslant b$)。
- Aeq 和 beq 分别为代表等式约束条件的矩阵和向量($Aeq \cdot x = beq$)。
- lb 和 ub 代表向量的边界($lb \leqslant x$，$x \leqslant ub$)。

在这里，只需要输入问题的约束条件部分。例如，如果没有边界约束条件，则使用下面的语句：

```
[x fval]=patternsearch(@objfun, x0, A, b, Aeq, beq)
```

对于没有约束条件的输入参数值，使用空的中括号“[ ]”。例如，如果没有不等式约束条件，就使用下面的语句：

```
[x fval]=patternsearch(@objfun, x0, [ ], [ ], Aeq, beq, lb, ub)
```

3）增加输出参数

要得到更多关于模式搜索运行的性能信息，可以使用下面语句调用 patternsearch 函数：

```
[x fval exitflag output]=patternsearch(@objfun, x0)
```

除了 x 和 fval 外，还返回下面的输出参数：

- exitflag：整数，表示算法是否成功。
- output：结构，包含关于求解器的性能的信息。

**2.** 在命令行设置模式搜索参数

通过将 options 结构作为模式搜索函数 patternsearch 的输入参数结构，可以设置模式搜索工具中任何有效的参数值。使用的语句为

```
[x fval]=patternsearch(@fitnessfun, nvars, A, b, Aeq, beq, lb, ub, options)
```

如果问题是无约束条件的，在约束条件输入时必须使用空括号，例如：

```
[x fval]=patternsearch(@fitnessfun,nvars,[ ],[ ],[ ],[ ],[ ],[ ],options)
```

还可以使用 psoptimset 函数生成 options 结构。例如，使用语句

```
options=psoptimset
```

返回一个带缺省值的 options 参数结构。

```
options=
        TolMesh:          1.0000e-006
        TolX:             1.0000e-006
        TolFun:           1.0000e-006
        TolBind:          1.0000e-003
        MaxIteration:     '100 * numberofvariables'
        MaxFunEvals:      '2000 * numberofvariables'
```

```
MeshContraction:     0.5000
MeshExpansion:       2
MeshAccelerator:     'off'
MeshRotate:          'on'
InitialMeshSize:     1
ScaleMesh:           'on'
MaxMeshSize:         Inf
PollMethod:          'positivebasis2n'
CompletePoll:        'off'
PollingOrder:        'consecutive'
SearchMethod:        [ ]
CompleteSearch:      'off'
Display:             'final'
OutputFcns:          [ ]
PlotFcns:            [ ]
PlotInterval:        1
Cache:               'off'
CacheSize:           10000
CacheTol:            2.2204e-016
Vectorized:          'off'
```

如果没有输入 options，则函数 patternsearch 使用缺省值。

每一个参数值保存在 options 结构成员中，如 options.MeshExpansion，即通过在 options 后面输入成员的名字，可以显示任何参数值。例如，要显示模式搜索网格扩展因子，可输入

```
options.MeshExpansion
ans=
    2
```

如果要生成与缺省值不同的 options 结构，可以使用函数 psoptimset 来完成。例如，为了改变网格扩展因子的值为 3(来代替其缺省值 2)，可输入

```
options=psoptimset('MeshExpansion', 3)
```

这时生成的 options 结构除了 MeshExpansion 值为 3 外，其他参数值都为缺省值。

如果现在调用输入参数为 options 的 patternsearch，则模式搜索使用的网格扩展因子为 3。

如果再决定改变 options 结构的另一个成员的值，例如，要设置 PlotFcns 为 @psplotmeshsize，它是在每一次迭代中画出网格尺寸的绘图函数，可以使用下面的语句来调用 psoptimset 函数：

```
options=psoptimset(options, 'PlotFcns', @psplotmeshsize)
```

结果保存 options 结构的所有成员的当前值，而 PlotFcns 则被改变为@plotmeshsize。

注意：如果省略 options 作为输入参数，则 psoptimset 重新设置 MeshExpansion 为它

的缺省值2.0。

也可以用一个命令同时设置MeshExpansion和PlotFcns值：

```
options=psoptimset('MeshExpansion',3,'PlotFcns',@plotmeshsize)
```

**3.** 从模式搜索工具使用参数和问题

使用psoptimset来生成一个参数结构，可以在模式搜索工具设置参数的值，并在MATLAB工作空间输出这些参数值。其详细描述可参见前面的"输入、输出参数和问题"部分的内容。如果在模式搜索工具中输出缺省参数，则options结构的参数设置与由下面命令得到的参数设置相同：

```
options=psoptimset
```

但是，'Display'的缺省值被改变了：当由psoptimset函数生成时'Display'为'final'；而在模式搜索工具生成时'Display'为'none'。

也可以从模式搜索工具输出整个问题，并且可以从命令行运行它。

## 9.3.3 模式搜索举例

本节通过实例说明如何运用模式搜索算法来求解问题，如何设置模式搜索的参数等。

**1.** 表决方法

在每一次迭代时，模式搜索在当前网格中画出这些点，也就是说，它计算目标函数在这些网格点的值，看哪些点的函数值比当前点的函数值小。在前面介绍的内容中，已经多次涉及模式搜索的工作过程，也列举了选举过程的例子。在模式搜索中，所定义的网格可以通过"Poll method(选举方法)"参数来指定，下面叙述其基本原理。

缺省的模式"Positive Basis(正基数)2N"由2N个向量组成，这里的N是目标函数中独立变量的个数。这2N个向量是：

[1 0 0 … 0]

[0 1 0 … 0]

…

[0 0 0… 1]

[-1 0 0…0]

[0 -1 0… 0]

…

[0 0 0… -1]

例如，目标函数有三个独立变量，则模式由以下6个向量组成：

[1 0 0]

[0 1 0]

[0 0 1]

[-1 0 0]

[0 -1 0]

[0 0 -1]

另外，如果设置"Poll method"为"Positive Basis NP1"，则模式由下列N + 1个向量组成：

[1 0 0 … 0]

[0 1 0 … 0 ]

…

[0 0 0 … 1]

[−1 −1 −1 … −1]

例如，如果目标函数有3个独立变量，则模式由下列4个向量组成：

[1 0 0]

[0 1 0]

[0 0 1]

[−1 −1 −1]

模式搜索有时使用“Positive Basis NP1”作为选举方法运行，速度较快，因为算法在每一迭代中搜索的点较少。例如：运行模式搜索来求解前面描述的有约束问题时，算法使用缺省选举方法“Positive Basis 2N”将执行2080次函数估计，但使用正基数NP1将只执行1413次函数估计。

然而，如果目标函数有许多局部最小值，使用“Positive Basis 2N”作为检测方法有可能避免找到的是局部最小值即非全局最小值，因为这个搜索的每一迭代在当前点周围搜索了更多的点。

**2.** 完全表决

按照缺省设置，模式搜索找到一个改进目标函数值的网格点时，算法就停止检测，并设置这个点作为下一次迭代的当前点。当这种情况发生时，许多网格点并没有得到检测，这些没有被检测的点中的一些点可能具有比模式搜索最先找到的那个点更小的目标函数值。

对于一些有局部最小值的问题，有时执行的是使模式搜索在每一次迭代中检测所有网格点，挑选一个具有最佳目标函数值的点，这就使得模式搜索在每一次迭代中需要搜索更多的点，以避免潜在的局部最小值而非全局最小值的问题。要使模式搜索检测整个网格，则需在“Poll”参数中设置“Complete poll”为On。

下面举例说明在模式搜索中如何使用完全表决。本例使用的目标函数如下：

$$f(x_1, x_2) = \begin{cases} x_1^2 + x_2^2 - 25 & x_1^2 + x_2^2 \leqslant 25 \\ x_1^2 + (x_2 - 9)^2 - 16 & x_1^2 + (x_2 - 9)^2 \leqslant 16 \\ 0 & \text{其他} \end{cases}$$

图9.22显示了这个函数的图形。函数的全局最小值在(0，0)处，其值是−25。然而，函数在(0，9)处有一局部最小值−16。

建立一个M文件来计算这个函数，复制和粘贴下列代码到MATLAB编辑器中的新M文件中：

```
function z=poll_example(x)
if x(1)^2 + x(2)^2 <=25
    z=x(1)^2 + x(2)^2 - 25;
elseif x(1)^2 + (x(2) - 9)^2 <=16
    z=x(1)^2 + (x(2) - 9)^2 - 16;
```

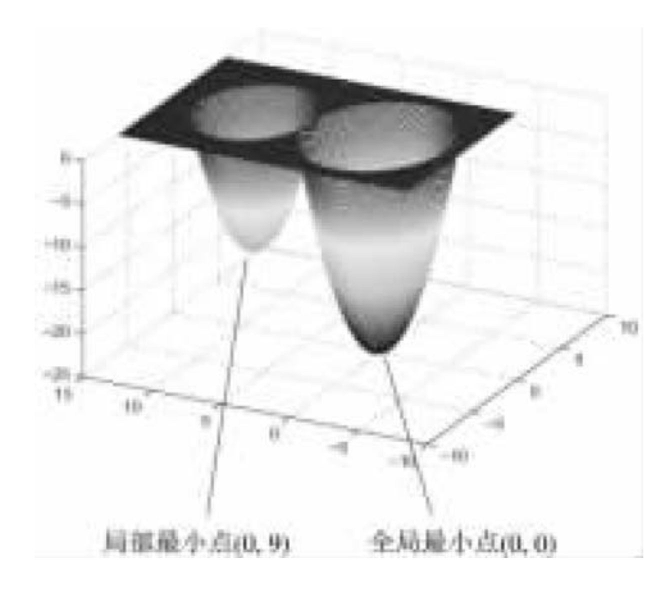



图 9.22　目标函数图形

```
else
    z=0;
end
```

保存该 M 文件在 MATLAB 指定的一目录中，文件名为 poll_example.m。

为了对函数运行模式搜索，在模式搜索工具中输入以下内容：

(1) 设置“Objective function”为 @poll_example。

(2) 设置“Start point”为[0，5]。

(3) 在“Display to command window(显示到命令窗口)”参数中设置“Level of display (显示级别)”为 Iterative。

使用“Complete poll”的缺省值 Off，单击“Start”按钮，运行模式搜索，模式搜索工具在“Status and results”窗口显示状态与结果信息，如图 9.23 所示。

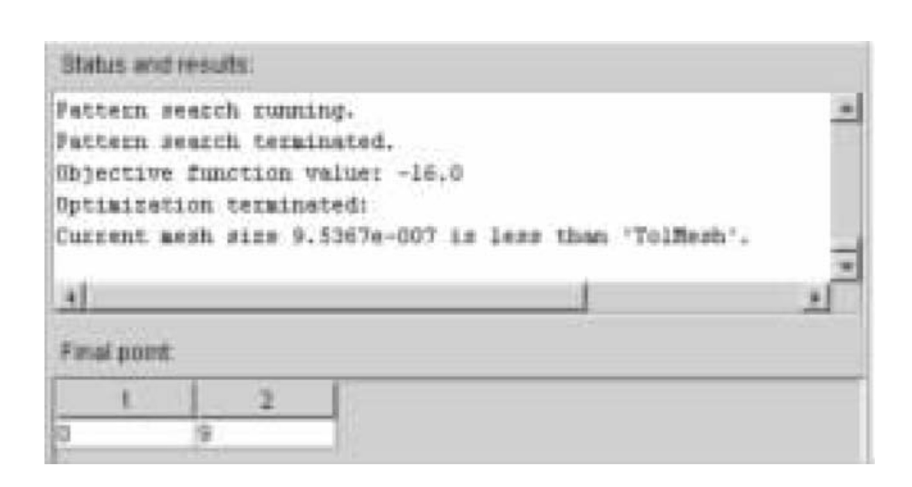



图 9.23　完全表决为 Off 时函数 poll_example 的状态与结果

从运行结果可以看出，模式搜索返回了在点(0，9)处的局部最小值。

在起始点处，目标函数值是 0，在第一次迭代中，算法检测如下网格点：

f((0，5)+(1，0))=f(1，5)=0

f((0，5)+(0，1))=f(0，6)=−7

一旦算法检测网格点(0，6)，这点的目标函数值比起始点的值要小，就会立即停止当前网格的检测，并设置下一代的当前网格点为(0，6)。在第一次迭代时，这个搜索朝局部最小点(0，9)移动。在命令行显示的头两行可看到如下内容：

| Iter | f-count | MeshSize | f(x) | Method |
|---|---|---|---|---|
| 0 | 1 | 1 | 0 | Start iterations |
| 1 | 3 | 2 | －7 | Successful Poll |

注意：模式搜索在第一次迭代中只执行了两次目标函数计算，增加总的函数计数从1到3。

设置“Complete poll”为On，并单击“Start”按钮，在“Status and results”窗口显示的结果如图9.24所示。

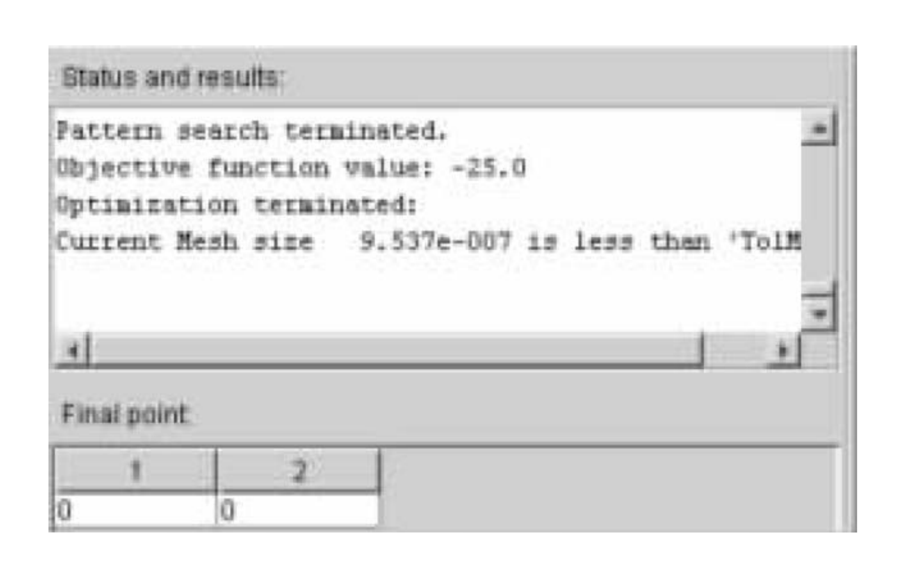



图9.24 完全表决为On时函数poll_example的状态与结果

这时，模式搜索找到全局最小值在(0，0)处。这一次和前一次不同的是“Complete poll”设置为On，模式搜索在第一次迭代中检测了所有4个网格点。

f((0，5) ＋ (1，0))＝f(1，5)＝0

f((0，5) ＋ (0，1))＝f(0，6)＝－7

f((0，5) ＋ (－1，0))＝f(－1，5)＝0

f((0，5) ＋ (0，－1))＝f(0，4)＝－9

由于最后一个网格点具有最小目标函数值，模式搜索选择它作为下一次迭代的当前点。在命令行显示的头两行可看到如下内容：

| Iter | f-count | MeshSize | f(x) | Method |
|---|---|---|---|---|
| 0 | 1 | 1 | 0 | Start iterations |
| 1 | 5 | 2 | －9 | Successful Poll |

在这种情况下，目标函数在第一次迭代中被计算了4次。结果，模式搜索朝全局最小点(0，0)移动。

图9.25是使用“Complete poll”参数选项分别为On和Off时返回的点序列的比较。

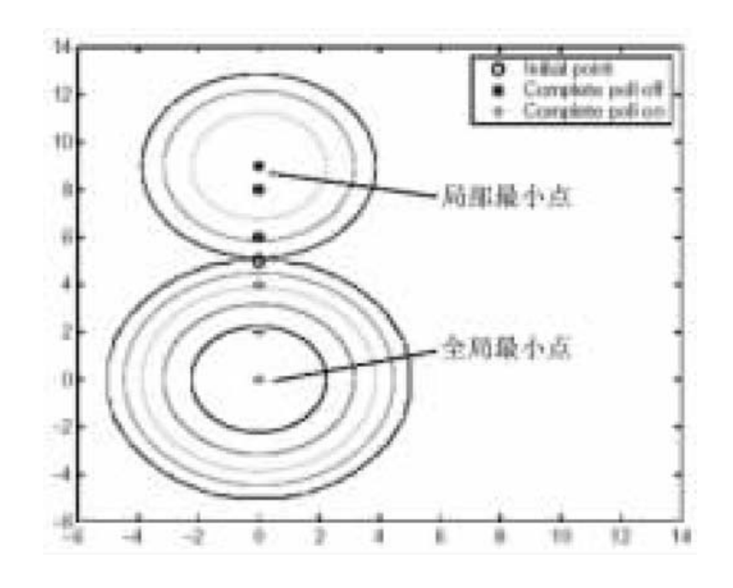



图 9.25　完全表决为 On 和 Off 时返回的点序列比较

**3．**使用搜索方法

对网格点的检测，模式搜索算法可在每一次迭代执行一可选步骤，称为 search。在每一次迭代的搜索步中，对当前点使用了另外一些优化方法。如果这个搜索步没有改进当前点，则检测步将执行。

下面举例说明对前面描述的有约束问题如何使用搜索方法。为了建立这个例子，首先定义起始点和约束条件。在 MATLAB 提示符下键入以下命令：

```
x0=[2 1 0 9 1 0];
Aineq=[-8 7 3 -4 9 0 ];
bineq=[7];
Aeq=[7 1 8 3 3 3; 5 0 5 1 5 8; 2 6 7 1 1 8; 1 0 0 0 0 0];
beq=[84 62 65 1];
```

然后，在模式搜索工具中，定义问题(输入目标函数和起始点)，输入约束条件，选择有关参数选项，如图 9.26 所示。

Objective function: @lincontest7
Start point: x0
Constraints:
Linear inequalities: A = Aineq　b = bineq
Linear equalities: Aeq = Aeq　beq = beq
Bounds: Lower =　Upper =
Plots
Plot interval: 1
☑ Best function value　☐ Mesh size
☑ Function count　☐ Best point
☐ Custom function:

图 9.26　设置模式搜索工具有关选项

为了比较，单击“Start”按钮，在这里先不使用搜索方法来运行这个例子，所显示的图形如图 9.27 所示。

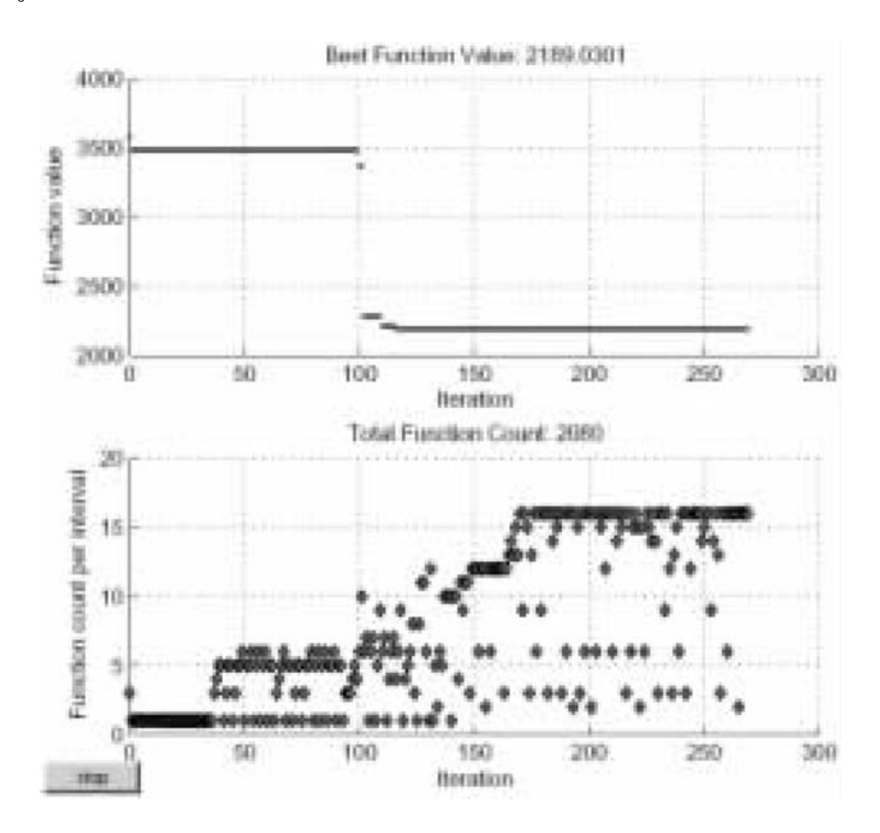



图 9.27　未使用搜索方法的运行结果

下面再看一下使用了搜索方法的效果。在“Search”参数的“Search method”字段选择“Positive Basis NP1”，这就对使用“Positive Basis NP1”模式的模式搜索设置了要使用搜索方法，随后单击“Start”按钮来运行算法，显示的图形如图 9.28 所示。

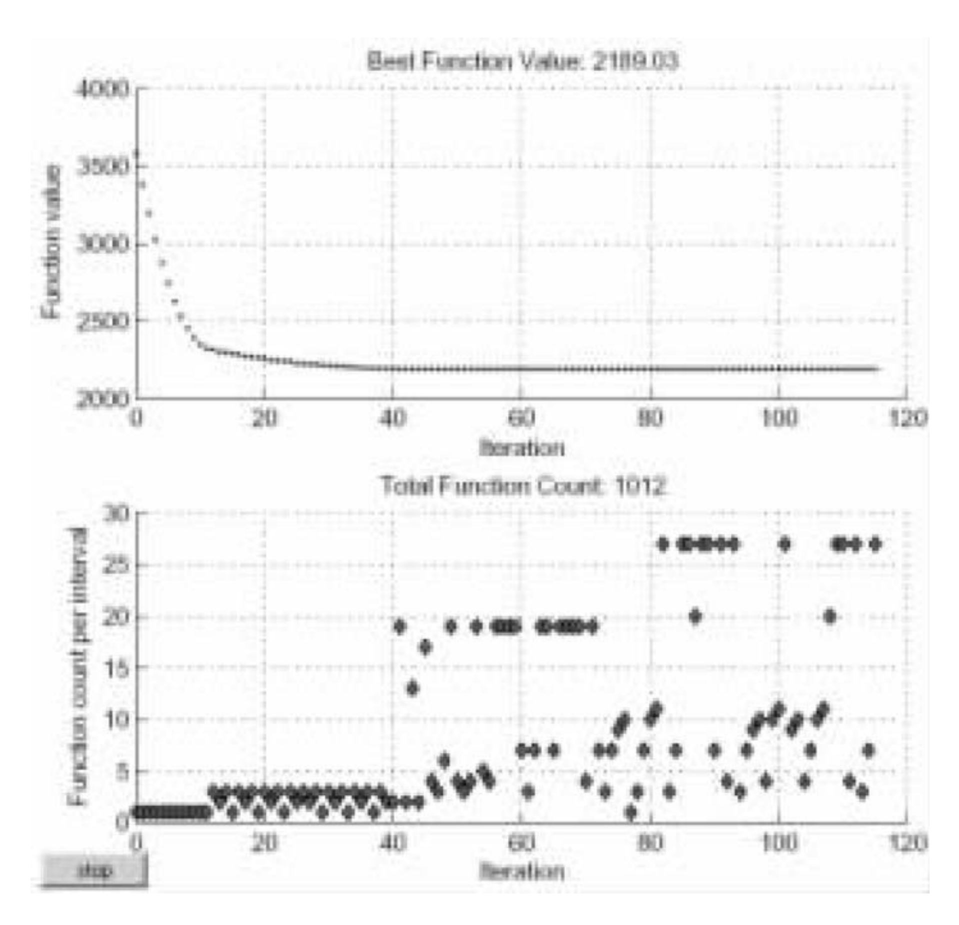



图 9.28　已使用搜索方法的运行结果

注意：使用搜索方法将减少总的函数计数（目标函数被估算的次数）接近50%，上例将迭代次数从270次减少到120次左右。

**4. 网格的扩展与收缩**

网格的扩展与收缩参数用于控制网格的大小在每一次迭代中是如何扩展或收缩的。“Expansion factor（扩展因子）”的缺省值是2.0，即模式搜索在每次成功的检测后，网格的大小扩展2倍。“Contraction factor（收缩因子）”的缺省值是0.5，即模式搜索在每次检测失败后，网格的大小收缩为原来的一半。

在模式搜索期间通过选择“Plots（绘图）”窗格中的“Mesh size”，就可以看到网格尺寸的扩展与收缩的图形显示，如图9.29所示。

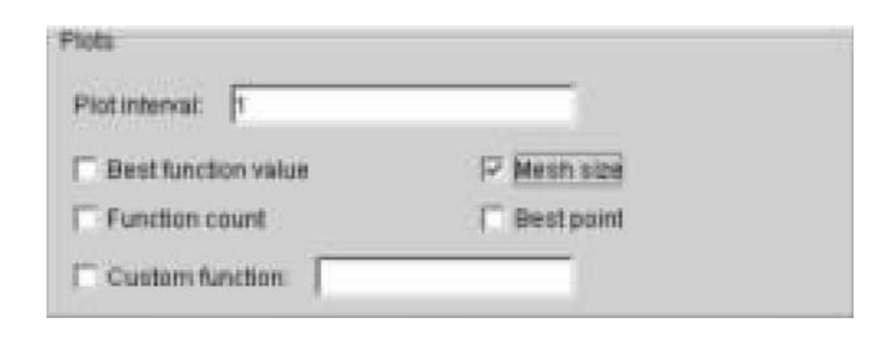



图9.29　绘图窗口

为了在命令行显示网格尺寸与目标函数值，在“Display to command window”参数中设置“Level of display（显示级别）”为Iterative，如图9.30所示。

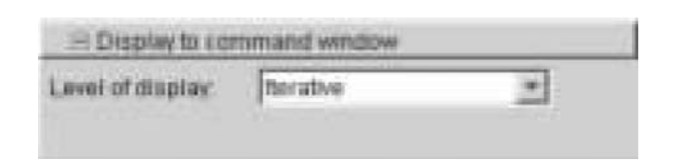



图9.30　设置显示级别为Iterative

当运行前面描述的求解有约束问题的例子时，模式搜索工具显示的结果如图9.31所示。

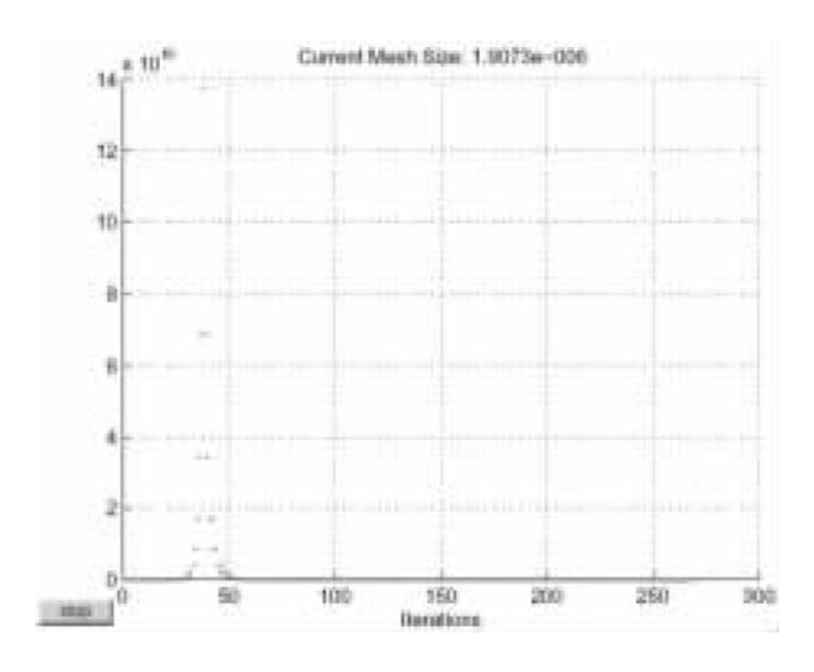



图9.31　有约束问题求解的网格尺寸

为了使看到的网格尺寸变化更加清晰，可改变算法中 Y 轴的刻度，方法如下：

（1）在绘图窗格的“Edit(编辑)”菜单中选择“Axes Properties(轴属性)”。

（2）在属性编辑器中，选择“Y”选项。

（3）设置“Scale(刻度)”为对数 Log。

图 9.32 显示了“Properties(属性)”编辑器中的设置。




图 9.32　属性编辑器的选择或设置

当单击“OK”按钮时，出现的图形显示如图 9.33 所示。

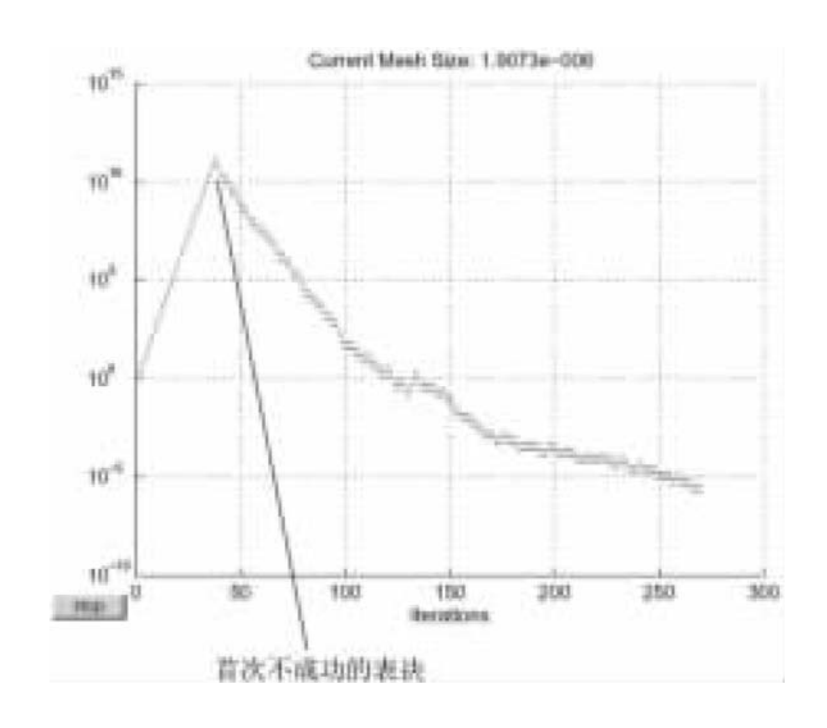



图 9.33　有约束问题求解的网格尺寸(对数刻度)

前 37 次迭代的结果是成功的检测，所以网格的大小在此过程中不断增加。通过查看在命令行显示的迭代输出信息，可知第一个不成功的检测发生在第 38 次迭代。

| Iter | f-count | MeshSize | f(x) | Method |
|---|---|---|---|---|
| 36 | 39 | 6.872e+010 | 3486 | Successful Poll |
| 37 | 40 | 1.374e+011 | 3486 | Successful Poll |
| 38 | 43 | 6.872e+010 | 3486 | Refine Mesh |

注意：由于在第 37 次迭代中检测是成功的，因而在下一次迭代中网格大小又扩大一倍，但在 38 次迭代中，检测不成功，因此网格大小缩小一半。

为了查看网格如何扩展和扩展后对模式搜索的影响，可做如下改变：

(1) 设置“Expansion factor”为 3.0。

(2) 设置“Contraction factor”为 0.75。

在“Mesh(网格)”窗格中，改变后的网格扩展因子和收缩因子如图 9.34 所示。

单击“Start”按钮，“Status and results”窗格显示的最终点与使用缺省的“Expansion factor”和“Contraction factor”设置近似相同，但模式搜索要花费更长时间才能达到这个点，如图 9.35 所示。

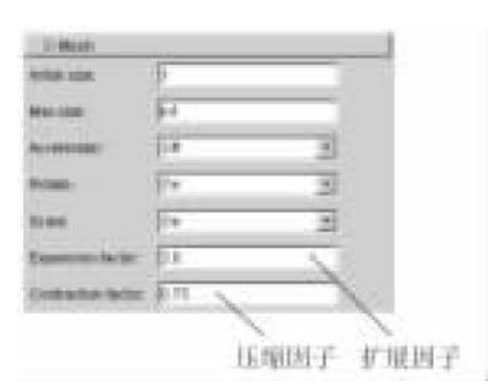



图 9.34　改变网格扩展因子和收缩因子

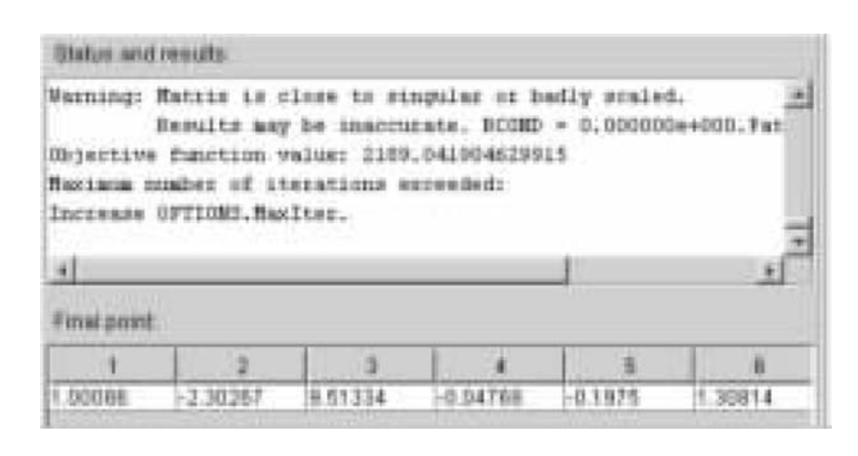



图 9.35　改变网格扩展和收缩因子后的结果

算法终止是因为它超过了最大的迭代次数，这个值是在“Stopping criteria(停止准则)”选项的“Max iteration(最大迭代)”字段中设置的。这个参数的缺省值是目标函数独立变量个数的 100 倍，例子中变量数是 6。

对算法改变 Y 轴的刻度，网格尺寸变化显示如图 9.36 所示。

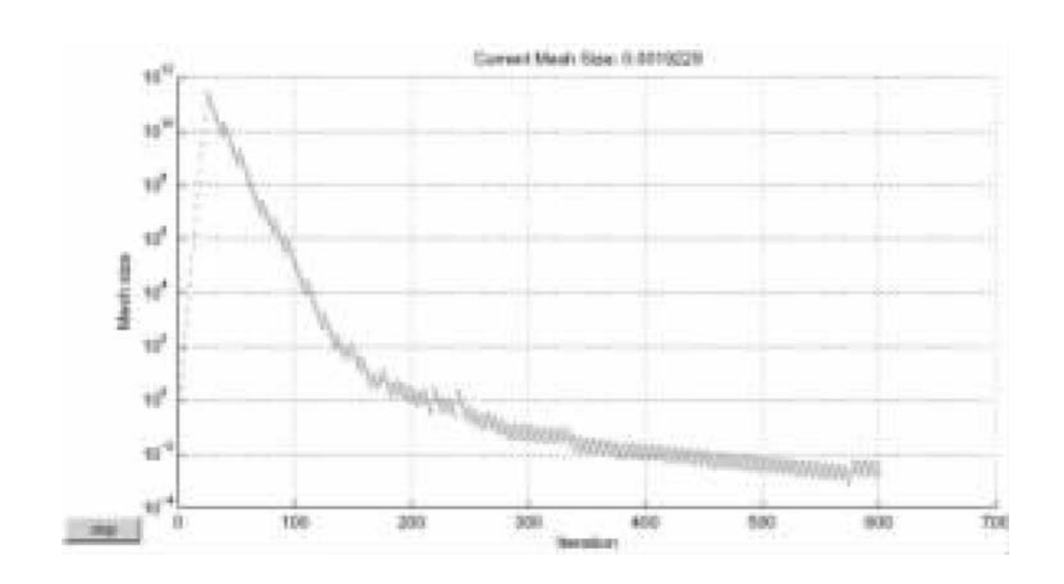



图 9.36　对数刻度的网格尺寸变化

注意：“Expansion factor”设置为 3.0，相对于缺省值 2.0 来说，网格尺寸增加更快；而“Contraction factor”设置为 0.75，相对于缺省值 0.5 来说，网格尺寸收缩更慢。

**5. 网格加速器**

网格加速器通过减少达到要求的网格容差所需的迭代次数，能使模式搜索更快地收敛到最佳点。当网格尺寸小于一定的值时，模式搜索使用一个小于“Mesh contraction”的因子收缩网格尺寸。

注意：强烈推荐只有对于当目标函数在最优点附近不太急剧变化或可能损失精度这类问题才使用网格加速器。对于那些可微的问题，也就是说，在解的附近其导数的绝对值不太大时才使用之。

为了使用网格加速器，需在“Mesh”选项中设置“Accelerator(加速器)”为 on。当运行前面描述的求解有约束问题的例子时，达到网格容差要求的迭代次数是 246，而“Accelerator”设置为 off 时，这个数为 270。迭代次数达到 200 次以后，比较网格的大小可看到网格加速器的作用，如图 9.37 所示。

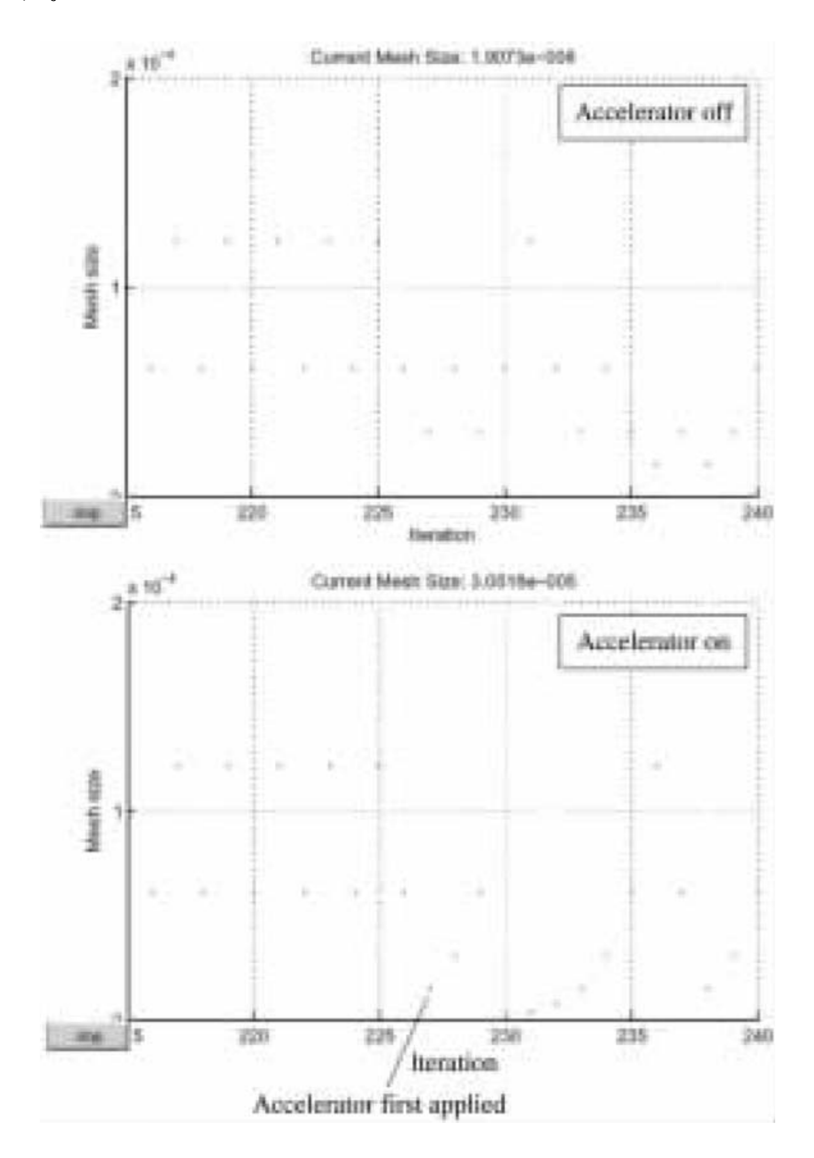



图 9.37 网格加速器的效果

在这两种情况下，网格的大小直到 226 次迭代都是相同的，不同是发生在 227 次迭代。将“Accelerator”设置为 off，迭代次数为 226 和 227 时，MATLAB 命令窗口显示下列信息：

| Iter | f-count | MeshSize | f(x) | Method |
|---|---|---|---|---|
| 226 | 1501 | 6.104e−005 | 2189 | Refine Mesh |
| 227 | 1516 | 3.052e−005 | 2189 | Refine Mesh |

注意：这里网格尺寸是乘以 0.5，即乘以缺省的“Contraction factor”。

将“Accelerator”设置为 on，迭代次数为 226 和 227 时，MATLAB 命令窗口显示下列信息：

| Iter | f-count | MeshSize | f(x) | Method |
|---|---|---|---|---|
| 226 | 1501 | 6.104e−005 | 2189 | Refine Mesh |
| 227 | 1516 | 1.526e−005 | 2189 | Refine Mesh |

注意：在此情形下，网格尺寸是乘以 0.25，即乘以比缺省的“Contraction factor”更小的收缩因子，从而使网格尺寸收缩得更快，起到一个加速的作用。

**6.** 使用缓存

典型情况下，模式搜索的任何给定迭代中的一些网格点可能与前面迭代中的点相符。对于缺省情况，尽管这些点已经计算过并且知道不是最优点，但模式搜索仍要重新计算这些网格点的目标函数值。如果计算目标函数值花费较长时间，譬如几分钟，这将使得模式搜索算法运行时间明显加长。

通过使用缓存可以消除这些冗余的计算，也就是说，需要把模式搜索已经检测过的历史点存储起来。为此，需要在“Cache(缓存)”参数中设置“Cache”为 on。在每次检测时，模式搜索查看当前网格点是否在缓存中某一点的规定“Tolerance(容差)”范围内，如果是，则搜索算法不再计算该点的目标函数值，而是使用在缓存中存储的函数值，并且移到下一点。

注意：当设置“Cache”为 on 时，模式搜索可能不能鉴别改进目标函数的当前网格中的某一点，因为它可能是在缓存的某一点的规定容差范围之内。为了得到结果，模式搜索在设置“Cache”为 on 时可能比设置“Cache”为 off 时运行更多的迭代次数。通常情况下，保持“Tolerance”为一个非常小的值总能获得较好的效果，尤其是对于那些高度非线性的目标函数而言更是如此。

为了说明这种情况，在“Plots”窗格中选择“Best function value(最佳函数值)”和“Function count(函数计数)”，并运行前面描述的求解有约束条件问题的例子，设置“Cache”为 off。在模式搜索完成时，所显示的图形如图 9.38 所示。这时，总的函数计数是 2080。

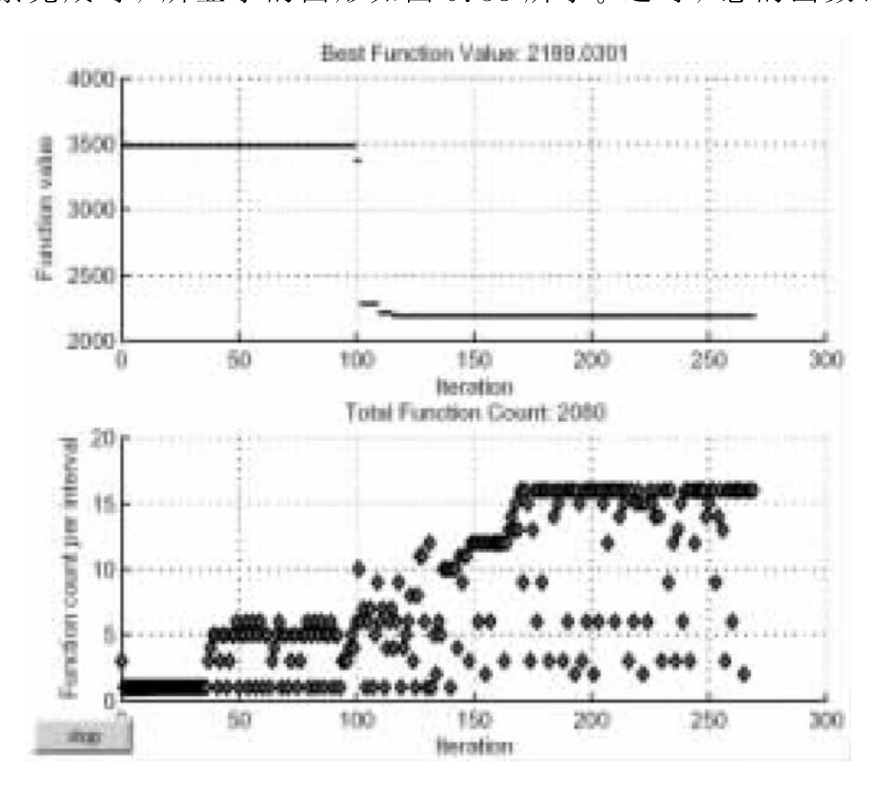



图 9.38　设置缓存为 off 时模式搜索的结果

现在设置“Cache”为 on，再次运行该例子，图形显示如图 9.39 所示。

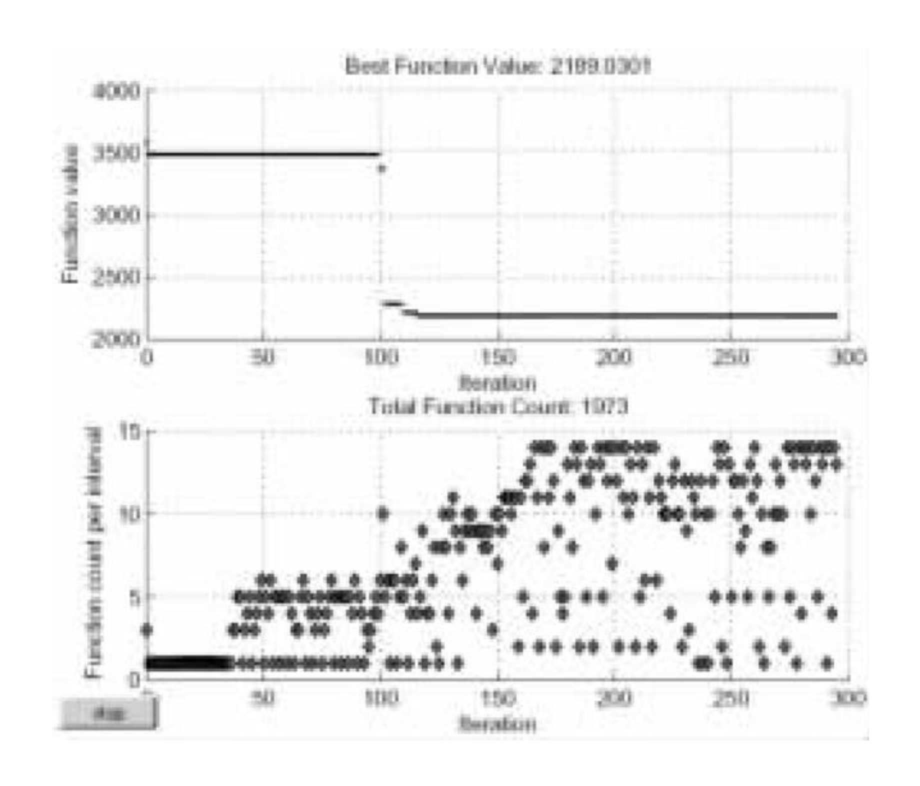



图 9.39　模式搜索工具显示结果

这时总的函数计数是 1973。可见，有效地减少了目标函数的计算次数。

**7. 设置求解器容差**

在搜索终止之前或以某种方法改变之前，“Tolerance”是一个非常小的参数，譬如一个网格尺寸。可以这样指定容差值：

(1)“Mesh tolerance(网格容差)”——当前网格尺寸小于这个值时，算法终止。

(2)“X tolerance(X 容差)”——在一次成功的检测后，从前一个最佳点到当前最佳点的距离小于这个值，则算法终止。

(3)“Function tolerance(函数容差)”——指明目标函数的最小容差。若目标函数在当前点的容差值小于“Function tolerance”，则算法停止，缺省值是 1e−6。

(4)“Bind tolerance(绑定容差)”——适用于约束问题，在一个线性约束被认定为激活以前，它指明如何接近一个点才一定能达到有效区域的边界。当线性约束有效时，模式搜索检测点的方向是平行于线性约束边界的那些网格点。

通常设置“绑定容差(Bind tolerance)”至少为“Mesh tolerance”、“X tolerance”和“Function tolerance”三者的最大者。

下面举例说明如何设置绑定容差。

本例说明“Bind tolerance”是如何影响模式搜索的，这个例子是查找下列函数的最小值：

$$f(x_1, x_2) = \sqrt{x_1^2 + x_2^2}$$

约束条件如下：

$$\begin{cases} -11x_1 + 10x_2 \leqslant 10 \\ 10x_1 - 10x_2 \leqslant 10 \end{cases}$$

注意：可使用函数 norm 计算目标函数值，这个问题的可行区域是图 9.40 中两条直线之间的区域。

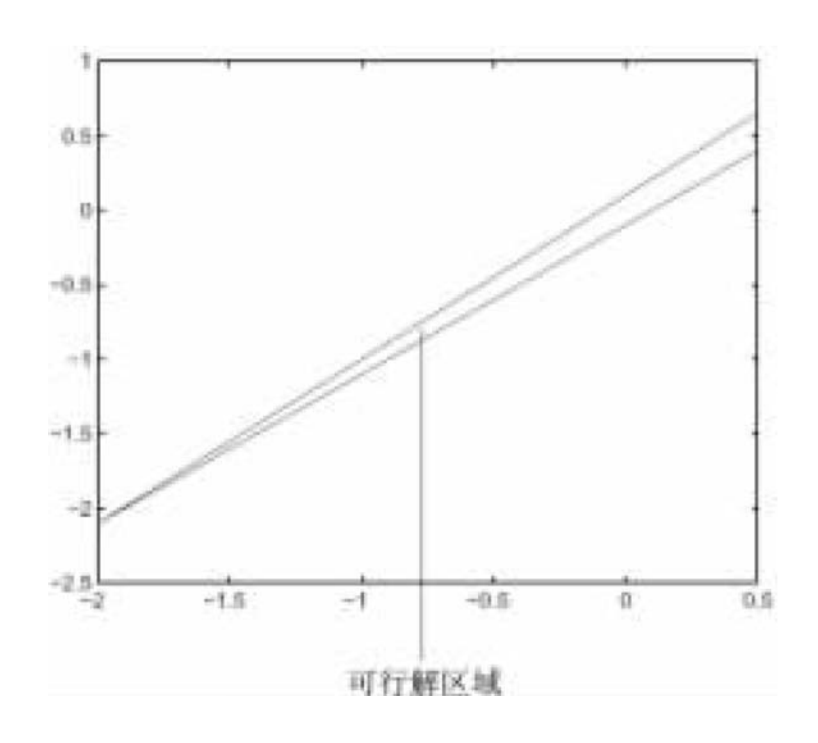



图 9.40　问题解的可行区域

1）使用缺省的绑定容差运行模式搜索

为了运行这个例子，键入 psearchtool，打开模式搜索工具，并输入以下信息：

（1）设置“Objective function”为 @(x) norm(x)。

（2）设置“Start point”为 [−1.001 −1.1]。

（3）在“Plots”窗格中选择“Mesh size”。

（4）在“Display to command window”参数中设置“Level of display”为 Iterative。

然后，单击“Start”按钮，运行模式搜索。

在 MATLAB 命令窗口中显示出来的前 4 次迭代检测是不成功的，这是由于网格点不在可行区域内。

| Iter | f-count | MeshSize | f(x) | Method |
|---|---|---|---|---|
| 0 | 1 | 1 | 1.487 | Start iterations |
| 1 | 1 | 0.5 | 1.487 | Refine Mesh |
| 2 | 1 | 0.25 | 1.487 | Refine Mesh |
| 3 | 1 | 0.125 | 1.487 | Refine Mesh |
| 4 | 1 | 0.0625 | 1.487 | Refine Mesh |

模式搜索在每一次迭代中收缩网格尺寸，直到有一点在可行区域之内为止。图 9.41 显示出在靠近边界上的初始点和第 5 次迭代的网格点。

最顶上的点是(−1.001，−1.0375)，它的目标函数值比起始点的目标函数值小，所以检测成功。因为从起始点到低界线的距离小于缺省的“Bind tolerance”值 0.0001，模式搜索将不考虑线性约束 $10x_1-10x_2\leqslant 10$ 是否满足，所以不在这个边界线平行线方向搜索点。

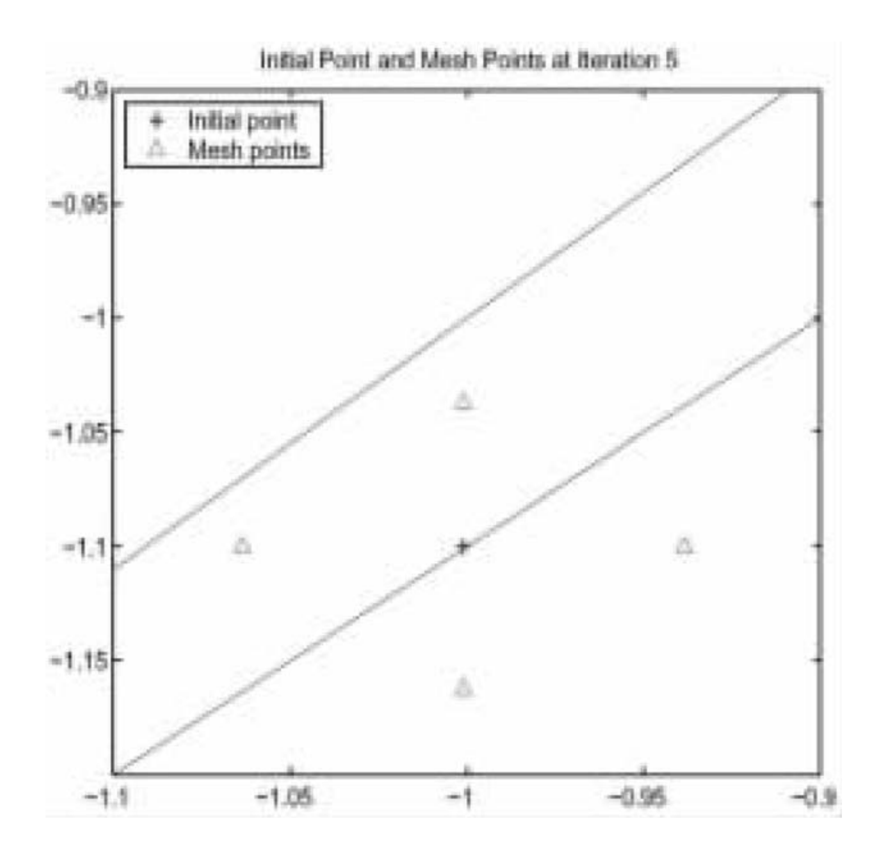



图 9.41　在靠近边界的初始点和第 5 次迭代的网格点

2）增大绑定容差的值

为了考察绑定容差的作用，改变“Bind tolerance”为 0.01，再次运行模式搜索。

这次，在 MATLAB 命令窗口中显示前两次迭代检测是成功的。

| Iter | f-count | MeshSize | f(x) | Method |
| --- | --- | --- | --- | --- |
| 0 | 1 | 1 | 1.487 | Start iterations |
| 1 | 2 | 2 | 0.7817 | Successful Poll |
| 2 | 3 | 4 | 0.6395 | Successful Poll |

因为从起始点到边界的距离小于“Bind tolerance”，第二个线性约束是有效的。在此情况下，模式搜索工具检测边界线 $10x_1-10x_2\leqslant 10$ 平行方向的点，结果是成功的。图 9.42 显示出起始点在边界平行线上有两个搜索点。

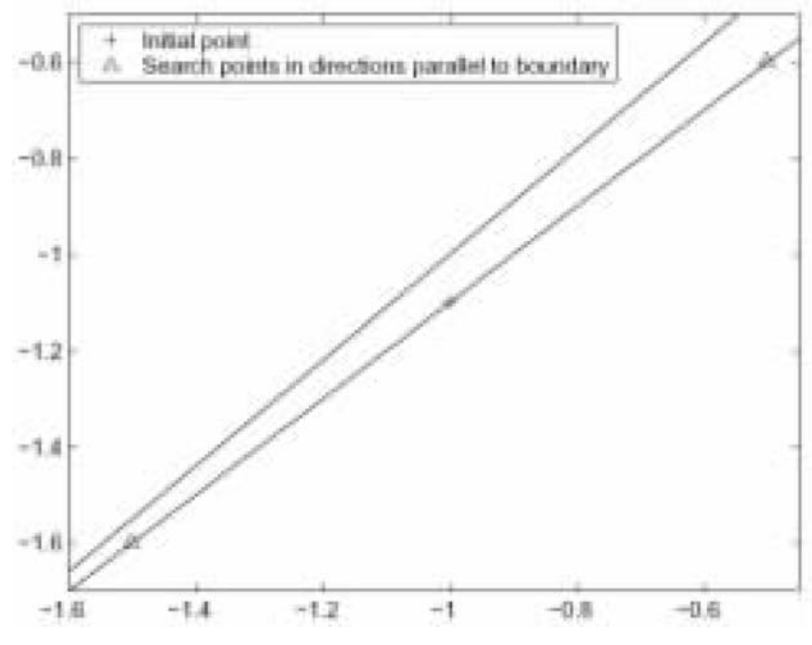



图 9.42　起始点在边界平行线上的两个搜索点

图 9.43 显示出对两种绑定容差的设置，模式搜索的前 20 次迭代点序列的比较情况。

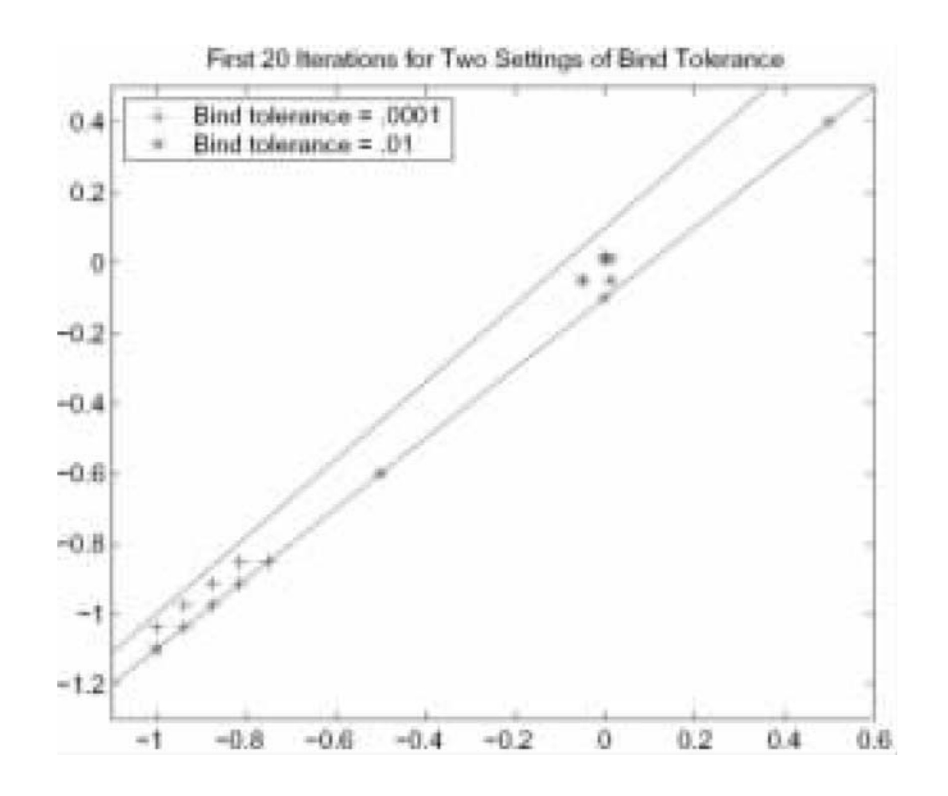



图 9.43　两种绑定容差设置下前 20 次迭代点序列比较

注意：当“Bind tolerance”是 0.01 时，搜索点向最优点的移动较快，这时模式搜索只需要 90 次迭代。而“Bind tolerance”是 0.0001 时，搜索要进行 124 次迭代。然而，当可行区域没有包含一非常精确的角度，就像前面的例子中，提高“Bind tolerance”将导致增加迭代次数，这是因为模式搜索将要检测更多的点。

### 9.3.4　参数化函数

有时，需要编写一个被 patternsearch 或者 ga 函数调用的函数，这个函数除了独立变量外，还有其他参数。例如，假设对于不同的变量 $a$、$b$、$c$，需要求下面方程的最小值：

$$f(x) = (a - bx_1^2 + x_1^{4/3})x_1^2 + x_1x_2 + (-c + cx_2^2)x_2^2$$

因为 patternsearch 和 ga 函数只接受依赖 $x$ 的目标函数、适应度函数，所以在调用 patternsearch 或者 ga 函数之前，必须将 $a$、$b$、$c$ 提供给函数。为此，下面给出两种方法：

（1）利用匿名函数参数化函数。

（2）利用嵌套函数参数化函数。

这里的例子将展示如何参数化一个目标函数，利用同样的方法，任何被 patternsearch 或者 ga 调用的用户函数，例如，为 patternsearch 定义的定制搜索方法或者为 ga 定义的定制尺度函数，都可以被参数化。

**1. 利用匿名函数参数化函数**

要参数化一个函数，首先编写一个 M 文件：

```
function y=parameterfun(x，a，b，c)
y=(a-b*x(1)^2+x(1)^4/3)*x(1)^2+x(1)*x(2)+(-c+c*x(2)^2)*x(2)^2;
```

在 MATLAB 路径的一个目录中保存这个 M 文件为 myfun.m。

现在，对于参数值 a＝4，b＝2.1，c＝4，求函数的最小值。

在 MATLAB 工作空间输入下列命令，给匿名函数定义一个句柄：

```
a＝4；b＝2.1；c＝4；
objfun＝@(x) parameterfun(x，a，b，c)；
x0＝[0.5 0.5]；
```

如果想使用模式搜索工具，则应设置“Objective function”为 objfun，设置“Start point”为 x0，如图 9.44 所示。




图 9.44　设置目标函数为匿名函数

然后，单击“Start”按钮，运行最优化，“Status and results”窗格中显示最后结果，如图 9.45 所示。

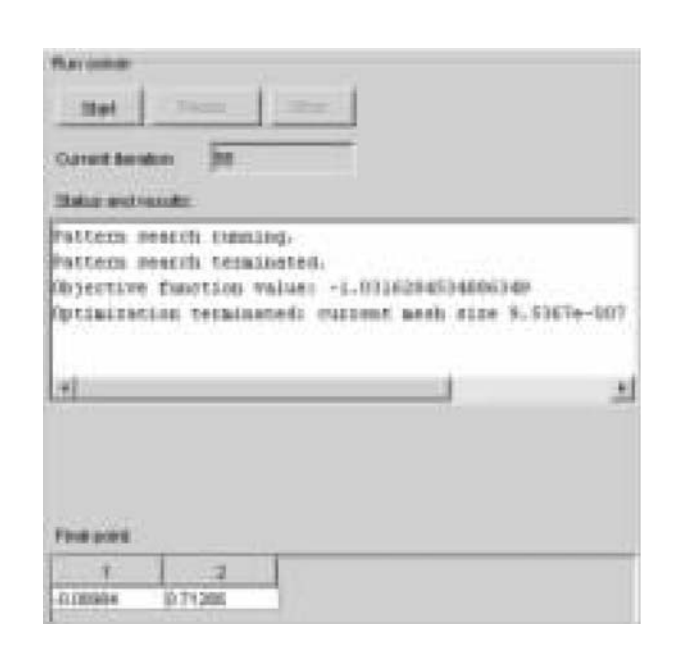



图 9.45　利用匿名函数参数化函数的结果

如果接着决定改变 a、b、c 的值，就必须重新建立一个匿名函数。例如：

```
a＝3.6；b＝2.4；c＝5；
objfun＝@(x) parameterfun(x，a，b，c)；
```

**2. 利用嵌套函数参数化函数**

参数化目标函数为匿名函数的另一个方法是：编写一个 M 文件，使得

(1) 接受输入的 a、b、c 和 x0。

(2) 将目标函数作为一个嵌套函数。

(3) 调用 patternsearch。

下面是该 M 文件的代码：

```
function [x fval]＝runps(a，b，c，x0)
[x，fval]＝patternsearch(@nestedfun，x0)；
    function y＝nestedfun(x)
```

```
        y=(a-b*x(1)^2+x(1)^4/3)*x(1)^2+x(1)*x(2)+(-c+c*
            x(2)^2)*x(2)^2;
    end
end
```

注意：目标函数在嵌套函数 nestedfun 内运算，它可访问变量 a、b、c。

为了运行最优化，可键入

```
[x fval]=runps(a, b, c, x0)
```

返回结果为：

```
Optimization terminated: current mesh size 9.5367e-007 is less than
'TolMesh'.
x=
    -0.0898 0.7127
fval=
    -1.0316
```

# 9.4 模式搜索参数和函数

本节详细说明 9 类模式搜索工具的参数选项和 4 个模式搜索函数。

## 9.4.1 模式搜索参数

首先描述模式搜索参数。设置模式搜索参数有两种方法，一种是使用模式搜索工具，另一种是在命令行调用模式搜索函数。

（1）如果使用模式搜索工具 psearchtool，则设置参数可从下拉列表中选择一参数或在一参数文本域中键入该参数的值。

（2）如果从命令行调用模式搜索函数 patternsearch，则首先要通过使用函数 psoptimset 来创建参数结构，进行参数设置，如下所示：

```
options=psoptimset('param1', value1, 'param2', value2, ...);
```

在这里，每一个选项参数均用两种方法给出：

（1）按其在模式搜索工具中的标签给出。

（2）按其在选项参数结构中的字段名给出。

例如：

（1）Poll method——模式搜索工具中的参数标签。

（2）PollMethod——在参数结构中对应的字段名。

参数选项分为以下 9 类：

（1）绘图参数。

（2）表决参数。

（3）搜索参数。

（4）网格参数。

（5）缓存参数。

（6）停止条件。

（7）输出函数参数。

（8）显示到命令窗口参数。

（9）向量参数。

**1．** 绘图参数

当模式搜索运行时，绘图参数可使来自模式搜索的数据画出图形。当选择绘图函数并运行模式搜索时，图形窗口在分开的轴上显示图形。在任意时间点可单击“Plots”窗格中的“Stop”按钮停止算法运行。

在绘图窗格中可以选择以下任意各项图形参数：

（1）“Plot interval(绘图间隔)”（PlotInterval）——指明相邻两次调用图形函数之间的迭代次数。

（2）“Best function value(最佳函数值)”(@psplotbestf)——画出最佳目标函数值。

（3）“Function count(函数计数)”(@psplotfuncount)——画出估算函数计数。

（4）“Mesh size(网格尺寸）”(@psplotmeshsize)——画出网格尺寸。

（5）“Best point(最佳点)”(@psplotbestx)——画出当前最佳点。

（6）“Custom(自定义)”——可以使用自己的绘图函数。在模式搜索工具中指定绘图函数的方法是：

- 选择“Custom function(自定义函数)”。
- 在文本框中键入@myfun，这里 myfun 是绘图函数名。绘图函数的结构在下面进行了详细的描述。

要在命令行调用 patternsearch 时显示图形，需设置参数选项 options 的 PlotFcns 字段为一个绘图函数的函数句柄。例如，要显示最佳函数值，设置参数如下：

```
options=psoptimset('PlotFcns', @psplotbestf);
```

要显示多个图形，语法如下：

```
options=psoptimset('PlotFcns', {@plotfun1, @plotfun2, ...});
```

这里@plotfun1，@plotfun2 等是命令行下绘图函数的名字。

下面介绍绘图函数的结构。绘图函数的第一行有如下形式：

```
function stop=plotfun(optimvalues, flag)
```

这个函数的输入变量是：

（1）optimvalues——包含解的当前状态有关信息的结构，这个结构包含以下字段：

- x——当前点。
- iteration——迭代次数。
- fval——目标函数值。
- meshsize——当前网格尺寸。
- funccount——估算函数计数。
- method——最后一次迭代的方法。
- TolFun——最后一次迭代的函数值的容差。
- TolX——最后一次迭代的 X 值的容差。

（2）flag——图形函数被调用的当前状态，这个标志值可能是：

- Init——初始状态。
- Iter——迭代状态。
- Done——最终状态。

这个函数的输出变量 stop，提供一种在当前迭代停止算法的方法，stop 可有以下值：

- False——算法继续下一次迭代。
- True——在当前迭代算法停止。

**2．**表决参数

表决参数用来控制模式搜索在每次迭代时怎样检测网格点。

“Poll method”(PollMethod)——指定算法用于创建网格的模式。有以下两个模式：

(1) 缺省模式“Positive basis 2N”，由以下 2N 个向量组成，N 是目标函数独立的变量个数：

[1 0 0 … 0]

[0 1 0 … 0]

…

[0 0 0 … 1]

[−1 0 0 … 0]

[0 −1 0 … 0]

…

[0 0 0 … −1]

例如，目标函数有 3 个独立变量，则模式由以下 6 个向量组成：

[1 0 0]

[0 1 0]

[0 0 1]

[−1 0 0]

[0 −1 0]

[0 0 −1]

(2)“Positive basis NP1”模式，由以下 N+1 个向量组成：

[1 0 0 … 0]

[0 1 0 … 0]

…

[0 0 0 … 1]

[−1 −1 −1 … −1]

例如，如果目标函数具有 3 个独立变量，则模式由下列 4 个向量组成：

[1 0 0]

[0 1 0]

[0 0 1]

[−1 −1 −1]

“Complete poll”(CompletePoll)——指明每次迭代时，当前网格中所有点均要进行 poll。“Complete poll”具有 On 或 Off 值。

(1) 如果设置“Complete poll”为 On，则算法将在每次迭代对当前网格中所有点进行检测，挑选具有最小目标函数值的点作为下一次迭代的当前点。

(2) 如果设置“Complete poll”为 Off(这是缺省值)，当算法一旦找到一个目标函数值比当前点小的点就停止检测。算法随后设置这个点为下一次迭代的当前点。

“Polling order(表决顺序)”(PollingOrder)——指明算法在当前网格中搜索的点的顺序。这个参数是：

(1) Random(随机的)——随机检测顺序。

(2) Success(成功的)——在每次迭代中，第一个搜索方向是上一次迭代中找到最佳点的方向，第一个点后，算法检测网格点采用与“Consecutive(连贯)”相同的顺序。

(3) Consecutive——算法检测网格点的顺序是“Consecutive order(连贯顺序)”，也就是在“Poll method”描述的模式向量顺序。

**3. 搜索参数**

搜索参数详细说明在每一次迭代检测之前算法能够执行的可选搜索项。如果搜索返回一个对目标函数有改进的点，则算法将这个点用于下一次迭代并省略检测。

“Complete search(完全搜索)”(CompleteSearch)——只应用于设置“Search method”选项为 Positive basis Np1、Positive basis 2N 或 Latin hypercube(拉丁超立方)的情形。“Complete search”取值为 On 或 Off。

对 Positive basis Np1 或 Positive basis 2N，Complete search 与表决参数中的Complete poll 有相同的意义。

“Search method”(SearchMethod)——指定搜索的方法，其选项有：

(1) None ([ ])——指明不搜索(缺省)。

(2) Positive basis Np1 ('PositiveBasisNp1')——指明模式搜索使用 Positive Basis Np1 作为“Poll method”。

(3) Positive basis 2N ('PositiveBasis2N')——指明模式搜索使用 Positive Basis 2N 作为“Poll method”。

(4) Genetic Algorithm(遗传算法)(@searchga)——指明搜索使用遗传算法，如果选择遗传算法，将有两个参数出现：

- Iteration limit(迭代限)——正整数，指明遗传算法作为模式搜索执行时迭代的次数。
- Options(参数)——遗传算法的参数结构，可用 gaoptimset 设置。

要在命令行改变“Iteration limit(迭代限)”和“Options”的缺省值，使用以下语法：

```
options=psoptimset('SearchMethod',{@searchga,iterlim,optionsGA})
```

这里 iterlim 是“Iteration limit”的值，而 optionsGA 是遗传算法的参数结构。

Latin hypercube(@searchlhs)——指明这种搜索的执行依赖于设置“Complete search”为 On 或 Off。

如果设置“Complete search”为 On，则算法将在每次迭代对由拉丁超立方搜索随机生成的所有点进行检测，选择一个具有最小目标函数值的点。

如果设置“Complete search”为 Off，则算法一旦有随机生成的比当前点函数值小的点，算法将立即停止检测，并将这个点传给下一次迭代。

如果选择 Latin hypercube，则有两个参数显示：

(1) Iteration limit——正整数，指明拉丁超立方搜索作为模式搜索执行时迭代的次数。

(2) Design level(设计级别)——正整数，指明设计级别，搜索点的数量等于“Design level”乘以目标函数独立变量的个数。

在命令行要改变“Iteration limit”和“Design level”的缺省值，可以使用以下语法：

```
options=psoptimset('SearchMethod',{@searchlhs,iterlim,level})
```

这里 iterlim 是“Iteration limit”的值，level 是“Design level”的值。

Nelder-Mead (@searchneldermead)——指明搜索使用 fminsearch，它是 Nelder-Mead 算法，如果选择 Nelder-Mead，将有两个参数显示：

(1) Iteration limit——正整数，指明 Nelder-Mead 算法作为模式搜索执行时所迭代的次数。

(2) Options——fminsearch 函数的参数结构，可用函数 optimset 创建。

要在命令行改变“Iteration limit”和“Options”的缺省值，可以使用以下语法：

```
options=psoptimset('SearchMethod',{@searchga,iterlim,optionsNM})
```

这里 iterlim 是“Iteration limit”值，optionsNM 是参数结构。

Custom——允许用户编写自己的搜索函数。使用模式搜索工具时，需要：

(1) 设置“Search function(搜索函数)”为 custom。

(2) 设置“Function name”为@myfun，这里 myfun 是函数名。

如果在命令行使用模式搜索函数 patternsearch，则可以用下列语句进行设置：

```
options=psoptimset('SearchMethod',@myfun);
```

如果要编写自己的搜索函数，可键入编辑命令：

```
edit searchfcntemplate
```

调出参考模板，利用参考模板来编写自己的搜索函数。

下面描述搜索函数的结构。搜索函数必须具有以下调用语法：

```
function [successSearch,nextIterate,FunEval]=
    searchfcntemplate(fun,x0,iterate,tol,A,L,U,...
    funeval,maxfun,searchoptions,objfcnarg)
```

搜索函数具有以下输入变量：

(1) fun——目标函数名。

(2) x0——起始点。

(3) iterate——迭代时的当前点，iterate 是一个结构，包含当前点及函数值。

(4) tol——容差，决定约束条件是否为活动的。

(5) A，L，U——定义可行区域。假设线性或边界条件为 $L \leqslant A * x \leqslant U$。

(6) Funeval——函数估算次数计数器，funeval 总是小于最大函数估算次数 Maxfun。

(7) Maxfun——函数估算次数的最大值。

(8) Searchoptions——设置搜索参数的结构，这个结构包含以下字段：

• Completesearch——如果值为“Off”，一旦找到一个较好的点，算法将终止，也就是不需要强迫减少充分条件。其缺省值为“On”，参见 psoptimset 对 completesearch 的描述。

• Meshsize——搜索步中所用的当前网格的大小。

• Iteration——当前迭代次数。

• Scale——刻度因子，用于计量设计点。

• Indineqcstr——不等式条件表示。

• Indeqcstr——等式条件表示。

• Problemtype——传递给搜索程序的标志，表示问题是'unconstrained'(无约束)、'boundconstraints'(边界约束)或'linearconstraints'(线性约束)。

• Notvectorized——一个标志，表示函数 fun 没有作为向量估算。

• Cache——使用缓存的标志，如果是“Off”，则没有使用缓存。

• Cachetol——用于缓存的容差，以确定两点是否相同。

• Cachelimit——缓存大小的上限。

(9) Objfunarg——目标函数的附加变量的单元数组。

搜索函数具有下列输出变量：

(1) Successsearch——一个布尔标识，表示搜索是否成功。

(2) Nextiterate——检测之后完成了成功的迭代，如果检测不成功，则下一迭代 nextiterate与当前迭代 iteration 相同。

注意：如果设置“Search method”为 Genetic algorithm 或 Nelder-Mead，那么推荐设置“Iteration limit”为缺省值 1，因执行搜索超过 1 次不大可能改进结果。

**4. 网格参数**

网格参数控制模式搜索使用的网格。以下参数是可利用的：

(1) Initial size(初始尺寸)(InitialMeshSize)——指明初始的网格尺寸，即从起始点到网格点的具有最短长度的向量。Initial size 是一正标量，缺省值是 1.0。

(2) Max size(最大值)(MaxMeshSize)——指明网格的最大值，当达到最大值后，在一次成功的迭代后，网格不再增大。网格最大值是一正标量，缺省值是 inf。

(3) Accelerator(加速器)(MeshAccelerator)——指明“Contraction factor(收缩因子)”在每次成功的迭代后是否乘以 0.5。“Accelerator”有两个值：On 和 Off，缺省值是 Off。

(4) Rotate(旋转)(MeshRotate)——指明当网格向量小于某一小值时是否乘以－1。“Rotate(旋转)”只能用于“Poll method”设置为 Positive basis Np1 且“Rotate”为缺省值为 On 时的情形。

注意：改变“Rotate”的设置不影响“Poll method”设置为 Positive basis 2N 时的检测。

(5) Scale (ScaleMesh)——指明算法是否使用模式向量乘以常数来对网格点进行比例变换。Scale 值可设置为 On 或 Off，缺省值为 On。

(6) Expansion factor(扩展因子)(MeshExpansion)——指明网格尺寸在一次成功的检测后增大的因子，缺省值是 2.0，即在一次成功检测后，使网格的大小乘以 2.0。“Expansion factor”必须是一正标量。

(7) Contraction factor (MeshContraction)——指明网格尺寸在一次不成功的检测后减小的因子，缺省值是 0.5，即在一次不成功检测后，使网格的大小乘以 0.5。“Contraction factor”必须是一正标量。

**5. 缓存参数**

模式搜索算法能保持已被检测过的点的一个记录，以使对同一个点的函数计算不会超过一次。如果目标函数要求一个相对较长时间的计算，则缓存参数就能提高算法运行的速度。为记录这些点所分配的内存区域，称为缓存。这个参数只能用于确定性函数，不能用于随机函数。

Cache (Cache)——指明是否使用缓存，它的值为 On 或 Off，缺省值是 Off。如果设置"Cache"为 On，则算法对网格中那些在缓存中的容差范围内的点不再评估目标函数。

Tolerance (CacheTol)——指明网格点相对于那些已在缓存中而算法省略对其进行检测的点来说有多近。"Tolerance"必须是一正的标量，缺省值是 eps。

Size (CacheSize)——指明缓存的大小，它是一正的标量，缺省值是 1e4。

**6. 停止条件**

停止条件决定什么情形导致模式搜索算法停止。模式搜索使用以下准则：

Mesh tolerance(网格容差)(TolMesh)——规定网格尺寸的最小容限，如果网格的大小小于"Mesh tolerance"，则算法停止。它的缺省值是 1e−6。

Max iteration(最大迭代次数)( MaxIteration)——指明算法执行的最大迭代次数。如果迭代次数达到"Max iteration"，则算法停止。它有下列选择：

(1) 100 * numberofvariables——最大迭代次数是目标函数独立变量数的 100 倍，这是缺省值。

(2) Specify(指定数)——指定一个正整数作为最大迭代次数。

Max function evaluations(最大函数估算)(MaxFunEval)——指明目标函数的最大估算次数，如果估算次数达到"Max function evaluations"，则算法停止。它可以有下列选择：

(1) 2000 * numberofvariables——最大函数估算次数是独立变量数的 2000 倍。

(2) Specify——指定一个正整数作为最大函数估算次数。

Bind tolerance (TolBind)——指明从当前点到可行区域边界距离的最小容差，绑定容差规定何时线性约束是活动的，它不是停止条件，其缺省值是 1e−3。

X tolerance(TolX)——指明相邻两次迭代的当前点之间的最小距离，如果相邻两点之间的距离小于"X tolerance"，则算法停止，其缺省值是 1e−6。

Function tolerance(TolFun)——指明目标函数的最小容差，如果目标函数在当前点的容差值小于"Function tolerance"，则算法停止，其缺省值是 1e−6。

**7. 输出函数参数**

输出函数是模式搜索算法在每次迭代时调用的函数，可以使用下列参数：

(1) 记录到新窗口(History to new window)(@psoutputhistory)——每间隔 Interval 的整数倍迭代次数时，在 MATLAB 命令行窗口中显示算法所计算的历史点。

(2) Custom——允许用户编写自己的输出函数。使用模式搜索工具指定输出函数：

- 选择"Custom function"。
- 在文本框中键入@myfun，这里 myfun 是自定义的函数名。

如果在命令行使用模式搜索函数 patternsearch，可以使用以下语句来设定输出函数：

```
options=psoptimset('OutputFcn', @myfun);
```

如果要编写自己的输出函数，可键入编辑器命令：

edit searchfcntemplate

调出参考模板，利用参考模板编写自己的输出函数。

下面叙述输出函数的详细结构。输出函数必须有如下的调用语法：

[stop,options,optchanged]=psoutputhistory(optimvalues,options,flag,interval)

输出函数有以下输入参数：

(1) optimvalues——包含关于解的当前状态信息的结构，这个结构包含以下字段：

- x——当前点。
- iteration——迭代次数。
- fval——目标函数值。
- meshsize——当前网格尺寸。
- funccount——函数估算次数。
- method——上一次迭代使用的方法。
- TolFun——上一次迭代函数值的容差。
- TolX——上一次迭代 X 值的容差。

(2) options——参数结构。

(3) flag——输出函数被调用时当前的状态，其值如下：

- init——初始状态。
- iter——迭代状态。
- done——最终状态。
- interval——任选的间隔变量。

输出函数返回以下变量：

(1) stop——提供停止算法当前迭代的一种方法，stop 有以下值：

- false——算法继续下一次迭代。
- true——算法在当前迭代终止。

(2) options——参数结构。

(3) optchanged——指示参数结构 options 是否改变的标志。

**8. 显示到命令窗口参数**

Level of display(显示级别)('Display')——指明模式搜索运行时有多少信息显示到命令行，可用参数是：

(1) Off ('off')——只显示最后的答案。

(2) Iterative ('iter')——显示每次迭代的信息。

(3) Diagnose ('diagnose')——显示每次迭代的信息。此外，还列出参数结构 options 中其缺省值已经被改变了的参数项。

(4) Final ('final')——模式搜索的结果(成功或不成功)、停止的原因、最终点。

Iterative 和 Diagnose 两者都显示以下信息：

(1) Iter——迭代次数。

(2) FunEval——累计的函数估算次数。

(3) MeshSize——当前网格尺寸。

（4）FunVal——当前点的目标函数值。

（5）Method——当前检测的结果。

Level of display 的缺省值如下：

（1）在模式搜索工具中是 Off。

（2）使用 psoptimset 创建的参数结构中是“final”。

**9．**向量参数

向量参数指明计算的目标函数是否是向量化的。

当设置“Objective function is vectorized（目标函数向量化）”为 Off（“Vectorize”为“off”）时，算法是在循环中计算网格点的目标函数值，每次通过迭代使用确定的一点调用目标函数。

另一方面，当设置“Objective function is vectorized”为 On（“Vectorize”为“on”）时，模式搜索算法一次调用目标函数计算所有网格点的目标函数值，这比在循环中调用计算它们更快些。无论怎样使用这个参数，目标函数都必须能接受任意行数的矩阵作为输入。

### 9.4.2 模式搜索函数

本节系统介绍模式搜索工具的四个函数的功能、格式和详细说明，这四个函数分别为 patternsearch、psearchtool、psoptimget 和 psoptimset。

**1．**函数 **patternsearch**

功能：使用模式搜索算法来搜索函数的最小值。

格式：

```
x=patternsearch(@fun, x0)
x=patternsearch(@fun, x0, A, b)
x=patternsearch(@fun, x0, A, b, Aeq, beq)
x=patternsearch(@fun, x0, A, b, Aeq, beq, lb, ub)
x=patternsearch(@fun, x0, A, b, Aeq, beq, lb, ub, options)
x=patternsearch(problem)
[x, fval]=patternsearch(@fun, x0, ...)
[x, fval, exitflag]=patternsearch(@fun, x0, ...)
[x, fval, exitflag, output]=patternsearch(@fun, x0, ...)
```

详细说明：函数 patternsearch 使用模式搜索算法来搜索目标函数的最小值。

x=patternsearch(@fun, x0)：用来解决那些形如$\min\limits_{x} f(x)$的无约束问题。在这里，fun 是一个 MATLAB 函数，它计算目标函数 $f(x)$的值。x0 是模式搜索算法的起始点。函数 patternsearch 可以接受形如@fun 的函数句柄作为目标函数，对目标函数返回一局部最小点 x。函数 fun 接受向量输入，返回一标量函数值。

x=patternsearch(@fun, x0, A, b)：查找函数 fun 的局部最小值点 x，服从线性不等式约束，约束的矩阵表示形式为：

$$A \cdot x \leqslant b$$

如果问题有 m 个线性不等式和 n 个变量，则 A 是一个 m * n 的矩阵，b 是一长度为 m 的向量。

x＝patternsearch(@fun，x0，A，b，Aeq，beq)：查找函数 fun 的局部最小值点 x，服从以下约束：

$$\begin{cases} A \cdot x \leqslant b \\ Aeq \cdot x = beq \end{cases}$$

这里 Aeq x＝beq 表示矩阵形式的线性等式约束，如果问题有 r 个线性等式约束，n 个变量，则 Aeq 是一 r×n 的矩阵，beq 是一长度为 r 的向量。

如果没有不等式条件，则 A 和 b 是空矩阵。

x＝patternsearch(@fun，x0，A，b，Aeq，beq，lb，ub)：对下列约束查找函数 fun 的局部最小值点 x：

$$\begin{cases} A \cdot x \leqslant b \\ Aeq \cdot x = beq \\ lb \leqslant x \leqslant ub \end{cases}$$

这里 lb 和 ub 分别表示变量的下边界和上边界。如果问题有 n 个变量，则 lb、ub 是一长度为 n 的向量。如果 lb 或 ub 为空(即没有提供)，它将自动分别扩展为－Inf 或 Inf。如果没有不等式或等式约束，则 A，b，Aeq，beq 将为空矩阵。

x＝patternsearch(@fun，x0，A，b，Aeq，beq，lb，ub，options)：查找函数 fun 的局部最小值点 x，使用参数结构 options 中的值代替缺省的优化参数，可以使用函数 psoptimset 来创建 options，A，b，Aeq，beq，lb，ub，options 为空矩阵时就使用缺省值。

x＝patternsearch(problem)：对问题 problem 查找最小值，problem 是一个结构，它具有以下字段：

(1) objective——目标函数。

(2) x0——开始点。

(3) Aineq——不等式约束矩阵。

(4) Bineq——不等式约束向量。

(5) Aeq——等式约束矩阵。

(6) Beq——等式约束向量。

(7) LB——x 的下界。

(8) UB——x 的上界。

(9) options——由函数 psoptimset 创建的参数结构。

(10) Randstate——可选参数，重新设置 rand 的状态。

(11) Randnstate——可选参数，重新设置 randn 的状态。

通过从模式搜索工具输出一个 problem，可以创建一结构 problem。可参见“9.3.1 浏览模式搜索工具”一节的“输入、输出参数与问题”的相关描述。

注意：问题结构 problem 必须有上面指明的所有字段。

[x，fval]＝patternsearch(@fun，x0，...)：返回目标函数在解 x 处的值 fval。

[x，fval，exitflag]＝patternsearch(@fun，x0，...)：返回 exitflag，用来描述退出函数 patternsearch 的条件，如果：

(1) exitflag＞0，patternsearch 收敛到一个解 x。

(2) exitflag＝0，patternsearch 达到最大的函数估算或迭代次数。

（3）exitflag<0，patternsearch 没有收敛到一个解。

[x，fval，exitflag，output]=patternsearch(@fun，x0，...)：返回一个包含有关搜索信息的结构输出 output。这个结构包含以下字段：

（1）function——目标函数。

（2）problemtype——问题的类型：无约束、边界约束或线性约束。

（3）pollmethod——表决方法。

（4）searchmethod——使用的搜索方法，可为任意。

（5）iteration——总的迭代次数。

（6）funccount——函数估算的总次数。

（7）meshsize——在 x 处网格的大小。

（8）message——算法停止的原因。

注意：函数 patternsearch 并不接受能输入复数类型数据的函数，为了解决复数解数据的问题，可编写一个能接受实数向量的函数，分开实数与虚数部分。

举例：给定下面的约束

$$\begin{bmatrix} 1 & 1 \\ -1 & 2 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leqslant \begin{bmatrix} 2 \\ 2 \\ 3 \end{bmatrix}$$

条件如下：

$$\begin{bmatrix} 0 \leqslant x_1 \\ 0 \leqslant x_2 \end{bmatrix}$$

这个问题的目标函数就是工具箱中提供的函数 lincontest6。为了查找最小值，相应的代码及运行结果如下：

```
A=[1 1；-1 2；2 1]；
b=[2；2；3]；
lb=zeros(2，1)；
[x，fval，exitflag]=patternsearch(@lincontest6，... [0 0]，A，b，[]，[]，lb)
Optimization terminated：
Next Mesh size (9.5367e-007) less than 'TolMesh.'


x=
     0.6667    1.3333
fval=
     -8.2222
exitflag=
     1
```

参见：psearchtool，psoptimget，psoptimset。

**2．函数 psearchtool**

功能：打开模式搜索工具。

格式：

psearchtool

详细说明：psearchtool 打开模式搜索工具，为模式搜索提供图形用户界面（参见图 9.13）。

可以使用模式搜索工具在优化问题上运行模式搜索算法并显示结果。

参见：patternsearch，psoptimget，psoptimset。

**3. 函数 psoptimget**

功能：得到模式搜索参数结构的值。

格式：

val＝psoptimget(options，'name')

详细说明：val＝psoptimget(options，'name')：从模式搜索参数结构 options 中返回参数 name 的值。

如果 name 的值在 options 中没有指定，则函数 psoptimget(options，'name')返回一空矩阵“[]”，只需要使用参数名中能惟一区分它的足够的前导字符即可。函数 psoptimget 忽略参数名中的大小写。

参见：psoptimset，patternsearch。

**4. 函数 psoptimset**

功能：创建一模式搜索参数结构。

格式：

options＝psoptimset

psoptimset

options＝psoptimset('param1'，value1，'param2'，value2，...)

options＝psoptimset(oldopts，'param1'，value1，...)

options＝psoptimset(oldopts，newopts)

详细说明：

options＝psoptimset(无输入参数)：创建一个参数结构 options，它包含模式搜索所需的参数，并设置参数为它们的缺省值。psoptimset 用没有输入或输出参数形式显示具有有效值的参数的完整列表。

options＝psoptimset('param1'，value1，'param2'，value2，...)：创建一参数结构 options，并设置'param1' 为 value1，'param2'为 value2 等等。任何没有指明的参数均被设置为它的缺省值。只需要使用参数名中能惟一区分它的足够的前导字符即可，忽略参数名中的大小写。

options＝psoptimset(oldopts，'param1'，value1，...)：创建 oldopts 的拷贝，修改指定参数'param1'的值为指定值 value1。

options＝psoptimset(oldopts，newopts)：用新的参数结构 newopts 组合已存在的参数结构 oldopts，在 newopts 中任意非空的参数覆盖 oldopts 中对应的参数。

参数：表 9.1 中列出了可以使用 psoptimset 设置的参数。对模式搜索参数和值的详细描述(参见 9.4.1 节“模式搜索参数”)。{}中的值是缺省值，也可在命令行键入 psoptimset 来查看优化参数和缺省参数。

表 9.1 用函数 **psoptimset** 设置的参数

| 参　　数 | 说　　明 | 值 |
| --- | --- | --- |
| Cache | 使 Cache 为“on”，检测时 patternsearch 保持一历史网格点，在子迭代时对已在其中的点不再检测。由于花费长时间计算目标函数会使 patternsearch 变慢，因此使用这个参数 | 'on' \| {'off'} |
| CacheSize | 缓存空间的大小 | 正标量\| {1e4} |
| CacheTol | 一个正的标量，说明当前点接近历史点达何种程度，才对它不进行检测 | 正标量\| {1e－10} |
| CompletePoll | 在当前迭代，完全检测 | 'on' \| {'off'} |
| CompleteSearch | 在当前迭代，完全检测 | 'on' \| {'off'} |
| Display | 显示级别 | 'off' \| 'iter' \| 'notify' \| 'diagnose'\| {'final'} |
| InitialMeshSize | 模式搜索的初始网格的大小 | 正标量\| {1.0} |
| MaxFunEvals | 目标函数估算的最大次数 | 正整数\|{2000 * 变量数} |
| MaxIteration | 最大迭代次数 | 正整数\|{100 * 变量数} |
| MaxMeshSize | 网格尺寸的最大值 | 正标量\| {Inf} |
| MeshAccelerator | 是否加速收到一最小值附近 | 'on' \| {'off'} |
| MeshContraction | 网格收缩因子，当迭代不成功时使用 | 正标量\| {0.5} |
| MeshExpansion | 网格扩展因子，当迭代成功时使用 | 正标量 \|{2.0} |
| OutputFcn | 指明在每次迭代时优化函数调用的自定义函数 | @psoutputhistory\|{none} |
| PlotFcn | 指明从模式搜索输出的绘图函数 | @psplotbestf \|<br>@psplotmeshsize\|<br>@psplotfuncount \| {[]} |
| PlotInterval | 指明相邻两次调用绘图函数相隔的迭代次数 | 正整数 |
| PollingOrder | 模式搜索的检测方向的顺序 | 'Random'\|'Success'\|<br>{'Consecutive'} |
| PollMethod | 模式搜索使用的检测策略 | {'PositiveBasis2N'}<br>\|'Positive BasisNp1' |
| ScaleMesh | 自动的变量尺度变换 | {'on'} \| 'off' |

| 参　　数 | 说　　明 | 值 |
| --- | --- | --- |
| SearchMethod | 模式搜索中使用的搜索方法的类型 | 'PositiveBasisNp1'<br>\|'PositiveBasis2N'\|<br>@searchga \|@searchlhs \|<br>@searchneldermead\| {[ ]} |
| TolBind | 绑定容差 | 正标量\| {1e－3} |
| TolCon | 约束容差 | 正标量\| {1e－6} |
| TolFun | 函数容差 | 正标量\| {1e－6} |
| TolMesh | 网格尺寸容差 | 正标量\| {1e－6} |
| TolX | 变量容差 | 正标量\| {1e－6} |
| Vectorized | 指定函数是否要向量化 | 'on' \| {'off'} |

参见：patternsearch，psoptimget。

# 参考文献


1 Goldberg D E. Genetic Algorithms in Search, Optimization and Machine Learning. Addison Wesley Publishing Company, 1989

2 Karr C L. Design of an Adaptive Fuzzy Logic Controller Using a Genetic Algorithm. Proc ICGA 4, 1991. 450～457

3 Holstien R B. Artificial Genetic Adaptation in Computer Control Systems, PhD Thesis, Depart. of Computer and Communication Sciences, University of Michigan, Ann Arbor, 1971

4 Caruana R A, Schaffer J D. Representation and Hidden Bias: Gray vs. Binary Coding. Proc. 6 th Int Conf Machine Learning, 1988. 153～161

5 Schmitendor W E, Shaw O, Benson R, et al. Using Genetic Algorithms for Controller Design: Simultaneous Stabilization and Eigenvalue Placement in a Region. Technical Report No. CS92－9, Dept of Computer Science, College of Engineering, University of New Mexico, 1992

6 Bramlette M F. Initialization, Mutation and Selection Methods in Genetic Algorithms for Function Optimization. Proc ICGA 4, 1991. 100～107

7 Lucasius C B, Kateman G. Towards Solving Subset Selection Problems with the Aid of the Genetic Algorithm. In Parallel Problem Solving from Nature 2, Manner R and Manderick B(Eds), Amsterdam: North-Holland, 1992. 239～247

8 Wright A H. Genetic Algorithms for Real Parameter Optimization. In Foundations of Genetic Algorithms, Rawlins JE (Ed), Morgan Kaufmann, 1991. 205～218

9 Michalewicz Z. Genetic Algorithms ＋ Data Structures ＝ Evolution Programs. Springer Verlag, 1992

10 Back T, Hoffmeister F, Schwefel HP. Survey of Evolution Strategies. Proc ICGA 4, 1991. 2～10

11 Grefenstette J J. Incorporating Problem Specific Knowledge into Genetic Algorithms. In Genetic Algorithms and Simulated Annealing, Davis L(Ed), Morgan Kaufmann, 1987. 42～60

12 Whitley D, Mathias K, Fitzhorn P. Delta Coding: An Iterative Search Strategy for Genetic Algorithms. Proc ICGA 4, 1991. 77～84

13 De Jong. Analysis of the Behaviour of a Class of Genetic Adaptive Systems. PhD Thesis, Depart. of Computer and Communication Sciences, University of Michigan, Ann Arbor, 1975

14 Baker J E. Adaptive Selection Methods for Genetic Algorithms. Proc ICGA 1, 1985. 101～111

15 Baker J E. Reducing Bias And Inefficiency in the Selection Algorithm. Proc ICGA 2, 1987. 14～21

16 Booker L. Improving Search in Genetic Algorithms. In Genetic Algorithms and Simulated Annealing, Davis L(Ed), Morgan Kaufmann Publishers, 1987. 61～73

17 Spears W M, De Jong. An Analysis of Multi-Point Crossover. In Foundations of Genetic Algorithms, Rawlins J E(Ed), 1991. 301～315

18 Syswerda G. Uniform Crossover in Genetic Algorithms. Proc ICGA 3, 1989. 2～9

19 Spears W M, De Jong K A. On the Virtues of Parameterised Uniform Crossover. Proc ICGA 4, 1991. 230～236

20 Caruana R A, Eshelman L A, Schaffer J D. Representation and Hidden Bias II: Eliminating Defining Length Bias in Genetic Search Via Shuffle Crossover. In Eleventh International Joint Conference on Artificial Intelligence, Sridharan NS(Ed), Morgan Kaufmann Publishers, 1989. 750～755

21 Mühlenbein H, Schlierkamp-Voosen. Predictive Models for the Breeder Genetic Algorithm. Evolutionary Computation, 1993, 1(1):25～49

22 Furuya H, Haftka R T. Genetic Algorithms for Placing Actuators on Space Structures. Proc ICGA 5, 1993. 536～542

23 Janikow C Z, Michalewicz Z. An Experimental Comparison of Binary and Floating Point Representations in Genetic Algorithms. Proc ICGA 4, 1991. 31～36

24 Tate D M, Smith A E. Expected Allele Convergence and the Role of Mutation in Genetic Algorithms. Proc ICGA 5, 1993. 31～37

25 Davis L. Adapting Operator Probabilities in Genetic Algorithms. Proc ICGA 3, 1989. 61～69

26 Fogarty T C. Varying the Probability of Mutation in the Genetic Algorithm. Proc ICGA 3, 1989. 104～109

27 De Jong K A, Sarma J. Generation Gaps Revisited. In Foundations of Genetic Algorithms 2, Whitley L D(Ed), Morgan Kaufmann Publishers, 1993

28 Whitley D. The Genitor Algorithm and Selection Pressure: Why Rank-based Allocations of Reproductive Trials is Best. Proc ICGA 3, 1989. 116～121

29 Huang R, Fogarty T C. Adaptive Classification and Control-Rule Optimization Via a Learning Algorithm for Controlling a Dynamic System. Proc 30th Conf. Decision and Control, Brighton, England, 1991. 867～868

30 Fogarty T C. An Incremental Genetic Algorithm for Real-Time Learning. Proc 6th Int. Workshop on Machine Learning, 1989. 416～419

31 Goldberg D E, Richardson J. Genetic Algorithms with Sharing for Multimodal Function Optimization. Proc ICGA 2, 1987. 41～49

32 Mühlenbein H, Schomisch M, Born J. The Parallel Genetic Algorithm as a Function Optimizer. Parallel Computing, 1991(17): 619～632

33 Starkweather T, Whitley D, Mathias K. Optimization Using Distributed Genetic Algorithms. Proc Parallel Problem Solving From Nature 1, Lecture Notes in Computer Science, Springer-Verlag, 1990(496):176～185

34 Georges-Schleuter M. Comparison of Local Mating Strategies in Massively Parallel Genetic Algorithms. In Parallel Problem Solving from Nature 2, Männer R and Manderick B(Eds), Amsterdam: North-Holland, 1992. 553～562

35 陈国良,王熙法等. 遗传算法及其应用. 北京:北京人民邮电出版社,1996

36 李书全. 遗传算法性能分析及其应用研究:[学位论文]. 天津:天津大学,1998

37 阳军. 遗传算法用于优化计算的问题研究:[学位论文]. 天津:天津大学,1998

38 周明,孙树栋. 遗传算法原理及应用. 北京:国防工业出版社,1999

39 飞思科技产品研发中心编著. MATLAB 6.5 辅助优化计算与设计. 北京:电子工业出版社,2003

40 王小平,曹立明. 遗传算法——理论、应用与软件实现. 西安:西安交通大学出版社,2002

41 王丽薇. 遗传算法的研究与应用:[学位论文]. 哈尔滨:哈尔滨工业大学,1994

42 方海鹏. 遗传算法的理论研究及其应用:[学位论文]. 大连:大连理工大学,1999

43 郝翔. 遗传算法及其在规则优化中的应用研究:[学位论文]. 西安:西安交通大学,1997

44 刘勇,康立山. 非数值并行计算(第二册)——遗传算法. 北京:北京科学出版社,1995

45 张铃,张钹. 统计遗传算法. 软件学报,1997,8(5):335～344

46 余建坤,张文彬等. 遗传算法及其应用. 云南民族学院学报,2002,11(4):193～197

47 贾兆红,倪志伟. 改进型遗传算法及其在数据挖掘中的应用. 计算机应用,2002,22(9):31～33

48 周克民,胡云昌. 遗传算法计算效率的改进. 控制理论与应用,2002,19(5):812～813

49 扬俊安,陈怡,钟子发. 标准遗传算法的改进及其在信息战领域应用展望. 中国人民解放军电子工程学院学报,2002,21(3):41～44

50 User's Guide: Genetic Algorithm and Direct Search Toolbox for Use with MATLAB. The MathWorks, 2004